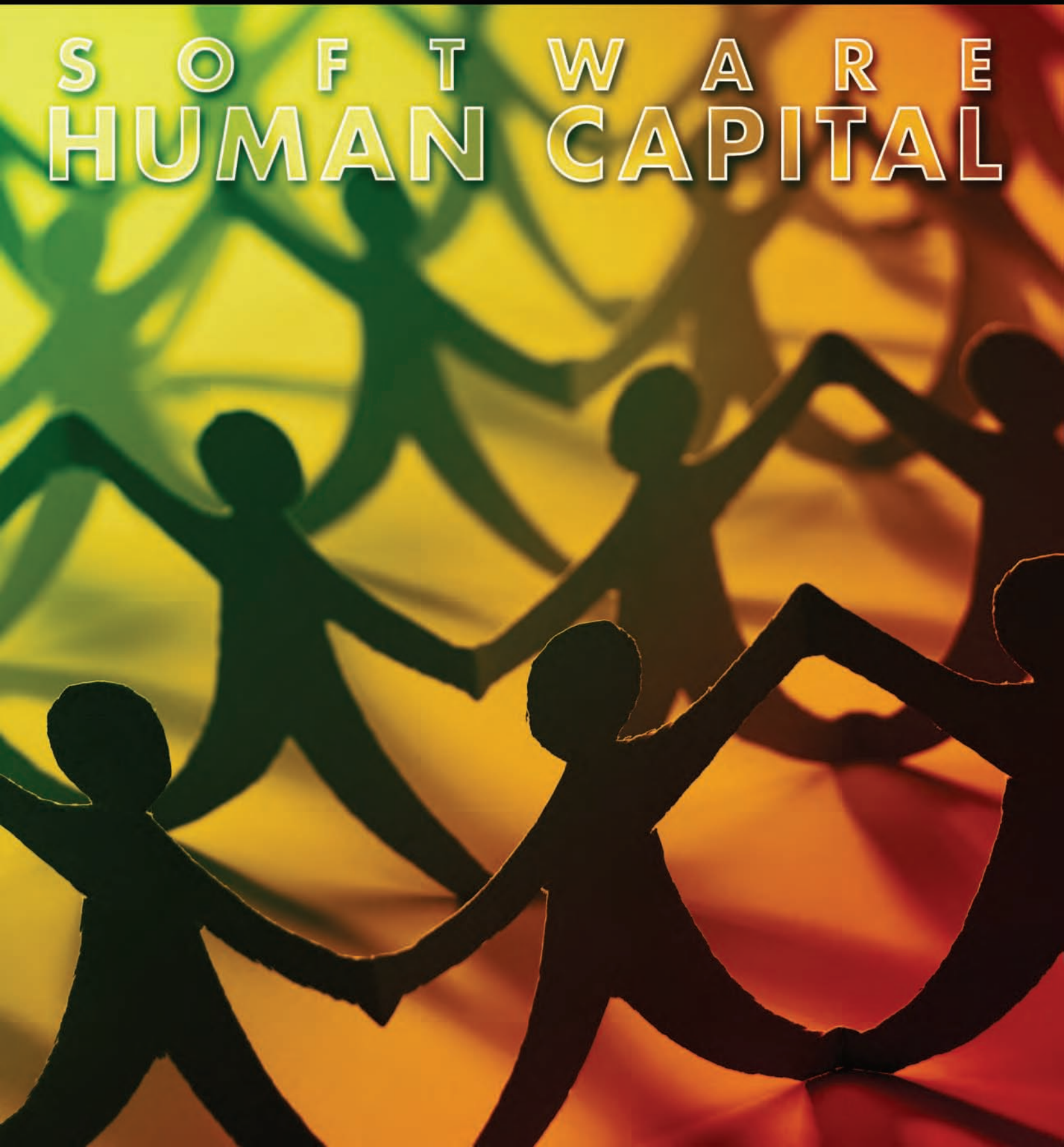
CROSSTALK

May/June 2010 The Journal of Defense Software Engineering Vol. 23 No. 3

SOFTWARE HUMAN CAPITAL



Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2010		**00-03-2010 to 00-06-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk. The Journal of Defense Software Engineering. Volume 23, Number 3, May/June 2010	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**32**	

Software Human Capital

Open Forum

ON THE COVER
Cover Design by Kent Bingham

Departments

CROSSTALK

OSD (AT&L) *Stephen P. Welby*
NAVAIR *Jeff Schwalb*
309 SMXG *Karl Rogers*
DHS *Joe Jarzombek*

MANAGING DIRECTOR *Brent Baxter*
PUBLISHER *Kasey Thompson*
MANAGING EDITOR *Drew Brown*
ASSOCIATE EDITOR *Chelene Fortier-Lozancich*
ARTICLE COORDINATOR *Marek Steed*
PHONE (801) 775-5555
E-MAIL stsc.customerservice@hill.af.mil
CROSSTALK ONLINE www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cybersecurity Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 17.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the author and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK *Online Services:* See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Developing Our Software Human Capital

The articles in this issue of CrossTalk discuss the human side of our software development processes, and are part of a very important dialogue. The DoD develops and delivers to our soldiers, sailors, marines, and airmen incredibly effective—but increasingly complex—weapons systems. Software has become such an integral part of these systems that it is virtually impossible to find a weapons system today that does not contain mission-critical software at its core.

CrossTalk would like to thank the Department of Defense Systems Engineering for sponsoring this issue.

As the complexity of our systems has increased, so has the need for effective systems and software engineering throughout the life-cycle. We face challenges in implementing robust system and software processes starting with requirements identification and analysis, through technology and architecture selection and assessment, analysis, and coordination of complex system design, development, and execution, to the delivery of rigorously tested production systems with a full complement of hardware and software capabilities. Our greatest challenges, however, may be in our approaches to building great people and teams: recruiting, growing, and maturing systems and software engineering professionals who will successfully deliver today and tomorrow's critical defense systems.

Current development programs are already challenged to find the highly skilled systems and software engineers we need, and numerous studies have raised concerns about our capability to meet our future human capital needs. The DoD is seeking to address this challenge in the near-term through improvements in the training, retention, and management of our workforce. New development methodologies, models, and tools offer promise in increasing the effectiveness and efficiency of our technical teams. Perhaps most importantly, we are continually looking at new ways to share our sense of excitement, purpose, and professional pride with the next generation of systems and software engineers.

The articles in this issue offer a range of viewpoints and thought-provoking insights about the human side of our software development and acquisition processes. They present a look at current challenges as well as innovative ideas while supporting a common theme: Managing the human side of our process is essential to delivering the best possible systems for the warfighter. I thank the authors for their ideas and hope readers find this issue of CrossTalk interesting and informative.

Stephen P. Welby
Director, Systems Engineering
Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics

Influencing Software Competencies Across the DoD Acquisition Workforce

Don Scott Lucero
Office of the Director of Defense Research and Engineering

The growing importance of software in delivering military capabilities to the warfighter increases the need for the DoD to identify and support software-specific competencies for the acquisition workforce. To help address this challenge, the Software Acquisition Training and Education Working Group (SATEWG) is defining competencies for each acquisition career field, and reviewing software acquisition curricula against these competencies. This article provides insight into the problems being addressed, the SATEWG's approach to these challenges, and their accomplishments to date.

The complexity and criticality of defense software poses a significant challenge to the acquisition workforce as well as the human capital experts who need to ensure that the workforce has the right competencies to deliver this essential capability to the warfighter. Add to this the task of identifying cross-functional software competencies that are critical for acquisition professionals, and you have the primary challenges facing the SATEWG.

There have been a number of initiatives aimed at improving DoD acquisition outcomes over the years, which have subsequently impacted the acquisition workforce. In the mid-'90s, the DoD adopted a policy encouraging the use of commercial products—rather than those developed to military specifications—in order to take advantage of the innovation available in the commercial marketplace. Commercial standards became preferred over military standards. The government moved toward specifying the expected performance of a system, rather than telling contractors how to build it.

In the '90s era of declining defense budgets, policymakers expected acquisition reform to bring about greater efficiencies in order to pay for defense acquisition. The Federal Acquisition Reform Act (FARA) of 1996 called for greater efficiencies in defense acquisition [1]. The FARA eliminated 15,000 members of the defense acquisition workforce, and called for reductions of 25 percent over the following five years. The focus of the defense acquisition workforce shifted ostensibly from engineering of systems to systems acquisition. The numbers of acquisition personnel dwindled and systems became larger and more complex, creating significant challenges for defense acquisition. These challenges were thrown into the spotlight by Government Accountability Office (GAO) annual audits [2].

The acquisition reform pendulum started swinging the other way when, in 2003, the DoD started an effort to reinvigorate systems engineering in defense acquisition. In 2009, the Secretary of Defense proposed hiring 20,000 new acquisition professionals by the year 2015 [3]. The Weapon Systems Acquisition Reform Act of 2009 established new Directors for Systems Engineering and Developmental Test and Evaluation and called for reports on these parts of the defense acquisition workforce [4]. The challenge, however, continues to be that software engineering is not currently designated by a standalone occupational career code, nor is it managed within the acquisition workforce as its own career field.

"The application of modern software technologies and the use of sound software engineering practices ... are important elements of program execution."

The evolution of acquisition policy has had a significant impact on the acquisition workforce and their ability to manage software acquisition.

Software-Specific Human Capital Challenges

Software is a unique and critical component in the products of DoD, and its reach extends across the acquisition career fields and each of the services at varying levels. The application of modern software technologies, and the use of sound software engineering practices over the acquisition life cycle, are important elements of program execution.

The DoD conducted the first phase of a software industrial base study in 2006 [5], finding that their dependence on larger, more complex software is increasing the risk of not delivering systems on schedule and within budget. Although the study found that the nation's overall number of software developers was adequate for the near-term, it found shortfalls in the number of top-tier software program managers, architects, and domain experts—with perhaps as few as 500 having the skills to develop the DoD's complex, software-intensive systems. Though the software industrial base study did not address the acquisition workforce per se, it is safe to say that these shortfalls in top-tier talent are evident there as well.

It should be noted that subsequent phases of the software industrial base study found shortfalls in the number of adequately trained software developers, which was the primary reason the Office of the Secretary of Defense (OSD) – Acquisition, Technology & Logistics (AT&L) sponsored development of a reference curriculum for graduate study of software engineering [6].

In [7], the National Defense Industrial Association (NDIA) recommends actions including the broadening of expertise "to enhance cross-functional and domain knowledge and skills." It is critical that the DoD begin identifying and embedding the basic software skills needed for each career field. This will reduce the reliance on software experts while increasing the overall abilities of the acquisition workforce.

In 2006, the Navy started the

Software Process Improvement Initiative (SPII), which identified and examined issues preventing software-intensive projects from meeting schedule, cost, and/or performance goals [8]. A survey conducted as part of the SPII effort found that:

- There is a lack of adequately educated and trained software acquisition professionals and systems engineers.
- There are no established education standards.
- Key staff experience levels are below average.

In [8], the SPII's Human Resources Focus Team recommended identifying the software acquisition training needs tailored to the respective roles and responsibilities for six acquisition career fields: program management, systems and software engineering, acquisition logistics, contracting, legal, and test and evaluation engineering. They also recommended that the DoD use the findings of the report as a baseline to analyze the software competencies and training of the acquisition workforce [8].

In February 2008, the DoD established the SATEWG to develop software competencies for the entire acquisition workforce—not just software experts—starting with program managers and systems engineers [9]. In addition, the SATEWG was chartered to develop and initiate a plan to address the gaps in the existing software acquisition curricula. The SATEWG is comprised of individuals from different organizations with the goal of promoting, across the DoD, collaboration focusing on software and human capital initiatives for the acquisition workforce.

SATEWG Membership

The SATEWG is comprised of representatives from organizations designated by the Under Secretary of Defense (AT&L), and others including the OSD, Army, Navy, Air Force, Defense Acquisition University (DAU), Air Force Institute of Technology, SEI, and Sevatec, Inc.

Each of these organizations plays a role in developing or supporting competencies and curricula, and their active participation has been critical to the success of the group. The diversity of these stakeholders has made for a stronger product. There are two types of SATEWG members: core team members and advisors. This structure encourages leadership involvement and provides flexibility for varying levels of commitment.

Developing a Software Competency Framework

A key challenge for the SATEWG was to identify aspects of software engineering that are truly unique to software and relevant to the broader acquisition workforce. For example, courses often address requirements management and configuration management, but they do not necessarily take into account the volatility of software requirements or the potential for spawning a multitude of slightly different software configurations.

Another challenge for the SATEWG was to identify aspects of software acquisition general enough to be considered critical by the broader acquisition workforce, yet specific enough to support building an interdisciplinary software skillset. This interdisciplinary software skill set reduces dependency on software experts, which in turn becomes more important as the acquisition workforce grows.

The SATEWG created an overarching body of skills called the software competency framework. It is used as the foundation for providing input to the competency models specific to each acquisition career field, and as a source for analyzing existing curricula. During the framework's development, the SATEWG reviewed 234 software competencies and 790 competency elements from the following sources:

- Existing DAU curricula.
- Competency studies and reports conducted by the services (e.g., SPII) [8].
- Industry best practices.
- Existing competency models such as the Software Engineering Body of Knowledge [10]; Systems Planning, Research, Development and Engineering (SPRDE); program management; and IT career fields.

The framework includes the competencies that are both unique to software and cross-functional in nature, so they can be generalized for the various acquisition career fields. Many software-related competencies, although important, weren't deemed by the SATEWG as different enough from the other disciplines to be included in the framework—at least from the perspective of the acquisition workforce. For example, software specifications are certainly different from typical system specifications; however, the process for managing these different types of specifications is quite

similar for the acquisition workforce.

The SATEWG also reviewed the persistent software development and acquisition issues to ensure that the competencies identified are relevant to the pressing needs. This review included the original 1968 NATO efforts defining software engineering [11] as well as the top software issues identified by the NDIA [12]. The most current source turned out to be a systemic analysis of software issues found in DoD reviews of acquisition programs [13].

Components of the Framework

The SATEWG framework consists of the following:

- **Knowledge Areas (4):** High-level descriptions of the overarching skills that make up the software elements of the job.
- **Competencies (29):** Definitions that provide information at a generalized level that allows flexibility for cross-functional comparison. Competencies describe the job requirements and individual capabilities at a broader, more process-oriented level than a single knowledge, skill, or ability. There are multiple competencies under each knowledge area.

The SATEWG decided not to identify the specific performance outcomes for each competency (i.e., the behavior[s] an employee must demonstrate for successful job performance). These expected outcomes will vary from career field to career field. Instead, the SATEWG decided that the performance outcomes should be defined by the groups that manage each career field.

The framework contains software knowledge areas and competencies within each knowledge area (see Table 1).

Applying the Framework

The SATEWG uses the software competency framework to work closely with each career field to help integrate software expertise into their existing competency models.

The SATEWG started working with the SPRDE expert panel to integrate software into their draft SPRDE competency model. The SATEWG identified the key competencies from a software perspective, while the SPRDE expert panel identified the key software competencies from their perspective. Using the competency framework, the SATEWG and SPRDE expert panel tailored the software competencies to the needs of the engineering workforce. The final SPRDE career field model now contains 13 elements that address software; more specifically, 14 of the framework competencies in Table 1 (marked with a "*") were mapped to the final SPRDE model.

The SATEWG followed a similar process for both the Test & Evaluation and Production, Quality & Manufacturing career fields. The software competency framework allows the SATEWG to provide input to the expert panels of each career field that is consistent—as well as customized—to the needs of each career field.

The SATEWG has also started the process of identifying gaps in the existing software acquisition curricula. To conduct this analysis, the SATEWG uses the software competency framework, as well as the DAU's terminal and enabling learning objectives from their software acquisition management courses.

Table 1: *SATEWG Software Competency Framework Summary*

Knowledge Area	Competencies
1. Software Acquisition and Sustainment Planning: The activities used to plan for the acquisition, development, and sustainment of software across the life cycle.	1. Software Impact on Acquisition Strategy* 2. Software Planning* 3. Software in the Work Breakdown Structure 4. Integrated Master Plan/Integrated Master Schedule 5. Planning for Software Transition and Sustainment
2. Software Development Considerations: Software development is a process of defining and executing software solutions from system-level requirements, which have been allocated to software. This includes the life-cycle activities such as designing, developing, integrating, and testing of the software components of a system. It also includes design considerations such as compatibility, extensibility, fault-tolerance, maintainability, packaging, reliability, reusability, security, and usability, as well as the development of associated documentation.	6. Software Architecture* 7. Software Requirements* 8. Integration of Software and Systems Engineering* 9. Software Design* 10. Software Development Methodology 11. Software Integration* 12. Software Interface Management 13. Software Modeling and Simulation 14. Verification & Validation of Software 15. Software in Systems Engineering Plans 16. Software Interoperability* 17. Software Safety* 18. Software Security* 19. System and Software Engineering Environment 20. Software Trade Studies
3. Software Management: Establishes a common framework for software life-cycle processes, with well-defined terminology that applies to the acquisition of systems and software products and services, to the supply, development, operation, maintenance, and disposal of software products, and to the software portion of a system.	21. Software Configuration and Data Management 22. Software Risk Management* 23. Software Technical Reviews 24. Software Quality Assurance* 25. Software Financial Management and Estimation* 26. Software Contracting Considerations 27. Software Measures*
4. Post-Deployment Software Support: The planning, sustainment, and management activities related to the performance of preventative, predictive, scheduled, and unscheduled actions aimed at maintaining or improving software performance (e.g., functionality, efficiency, reliability, availability, maintainability, security, and safety).	28. Transition to Sustainment 29. Sustainment

Future Direction

While the SATEWG remains focused on the goals outlined by the original charter, members are identifying opportunities that go beyond it. These efforts further bridge the gap between current and desired software proficiency and also reach a new audience: software experts who are critical in managing the complexity of today's software-intensive systems. Such efforts include:

- Formally validating the framework.
- Fostering a learning environment and addressing the training needs of software experts.
- Establishing a government-wide occupational career code for software engineering.

The SATEWG will start pursuing these additional efforts when the elements of the original charter are met. Current goals and future efforts will require support and collaboration with software and human capital leaders across the DoD. The SATEWG will continue to apply a collaborative approach to ensure continued success.

The SATEWG welcomes the involvement of software and human capital leaders across the DoD. Please contact the author if you would like to receive more information about the SATEWG's efforts.

Conclusion

Several studies conducted recently have highlighted both the human capital and software-related issues facing the DoD. To address the growing concern regarding software complexity and the capacity of the acquisition workforce, the SATEWG has made strides to ensure that software-related skills are both embedded in competency models and fostered within existing curricula.

The efforts of the SATEWG have led to the development of a framework

which lists the critical software competencies that are cross-functional and can be customized for each career field in the DoD. The SATEWG also uses this framework to review existing courses to ensure that the acquisition workforce is being trained in the necessary areas of software.◆

Acknowledgements

The author would like to thank the members of the SATEWG for their participation, feedback, and contributions. Your expertise, focus, and hard work are greatly appreciated. Also, thank you to Abby Fronk, Paul Kulgavin, and Kristy Murphy from Sevatec for all of your support and contributions to the SATEWG efforts and this article.

References

1. National Oceanic and Atmospheric Administration – Office of the Chief Information Officer and High Performance Computing and Communications. *Federal Acquisition Reform Act of 1996*. <www.cio.noaa.gov/Policy_Programs/fara.pdf>.
2. GAO. *Defense Acquisitions: Charting a Course for Lasting Reform*. Statement of Paul Francis, Managing Director Acquisition and Sourcing Management. GAO-09-663T. 30 Apr. 2009 <www.gao.gov/new.items/d09663t.pdf>.
3. GAO. *Department of Defense: Additional Actions and Data Are Needed to Effectively Manage and Oversee DoD's Acquisition Workforce*. GAO-09-342. Mar. 2009 <www.gao.gov/new.items/d09342.pdf>.
4. 111th Congress. *Weapon Systems Acquisition Reform Act of 2009*. Public Law 111–23. 22 May 2009 <www.acq.osd.mil/sse/docs/PUBLIC-LAW-111-23-22 MAY2009.pdf>.
5. Center for Strategic and International Studies Defense – Industrial Initiatives Group. *An Assessment of the National Security Software Industrial Base*. 19 Oct. 2006 <http://csis.org/files/media/csis/pubs/061019_softwareindustrialbase.pdf>.
6. Stevens Institute of Technology – School of Systems and Enterprises. *Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering*. Vers. 1.0. 30 Sept. 2009.
7. NDIA – Systems Engineering Division. *Report on Systemic Root Cause Analysis of Program Failures*. Dec. 2008 <www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/NDIASRCAReportFINA18Dec2008.pdf>.
8. Assistant Secretary of the Navy for Research, Development, and Acquisition. *Software Process Improvement Initiative Human Resources Focus Team: Role-Based Right-Fit Training Technical Report*. 6 Nov. 2007 <https://acc.dau.mil/GetAttachment.aspx?id=180970&pname=file&aid=31763&lang=en-US>.
9. Under Secretary of Defense for AT&L. *Establishment of Defense Acquisition Workforce Improvement Act Software Acquisition Training and Education Working Group*. 2008.
10. Hilburn, Thomas B., et al. *A Software Engineering Body of Knowledge Version 1.0*. SEI, Carnegie Mellon University. Technical Report. CMU/SEI-99-TR-004, ESC-TR-99-004. Apr. 1999 <www.sei.cmu.edu/reports/99t004.pdf>.
11. Naur, Peter, and Brian Randell, eds. *Software Engineering – Report on a Conference Sponsored by the NATO Science Committee*. Garmisch, Germany, 7-11 Oct. 1968. Jan. 1969 <http://homepages.cs.ncl.ac.uk/brian.randell/NATO /nato1968.pdf>.
12. NDIA – Systems Engineering Division. *Top Software Engineering Issues Within Department of Defense and Defense Industry*. Task Group Report. Vers. 5a. 26 Sept. 2006 <www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Content/ContentGroups/Divisions1/Systems_Engineering/PDFs18/NDIA_Top_SW_Issues_2006_Report_v5a_final.pdf>.
13. "Software Problems Found on DoD Acquisition Programs." Presentation to the Advisory Group for the Army's Strategic Software Improvement Program. Oct. 2008.

About the Author

Don Scott Lucero is the deputy director of software engineering in the Office of the Under Secretary of Defense (AT&L). He has bachelor's and master's degrees in computer science and is DoD acquisition-certified in four career fields.

1851 S Bell ST
#102
Arlington, VA 22202
Phone: (703) 602-0851
E-mail: scott.lucero@osd.mil

Coming Events

June 6-11
Association for Computing Machinery SIGMOD/PODs Conference
Indianapolis, IN
www.sigmod2010.org

June 27- July 1
Cisco Live! 2010
Las Vegas, NV
www.cisco-live.com

June 28-30
Command and Control Summit 2010
Washington, D.C.
www.C2Event.com

July 1-3
22nd International Conference on Software and Knowledge Engineering
Redwood City, CA
www.ksi.edu/seke/seke10.html

July 12-15
20th Anniversary INCOSE International Symposium 2010
Chicago, IL
www.incose.org/symp2010

July 25-29
34th Annual IEEE Computer Software and Applications Conference
Los Angeles, CA
www.siggraph.org/s2010

August 9-13
Agile Conference 2010
Nashville, TN
http://agile2010.agilealliance.org

August 15-19
30th International Cryptology Conference
Santa Barbara, CA
www.iacr.org/conferences/crypto2010

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: <marek.steed.ctr@hill.af.mil>.**

Recruiting Software Practitioners: The Importance of Self-Efficacy

Dr. Orit Hazzan
Israel Institute of Technology

Dr. Tali Seger
Ruppin College, Israel

Software organizations face challenges when trying to recruit highly competent software practitioners who can successfully participate in and contribute to a cooperative working environment. This article suggests—based on the presented research conducted in a large international communication company—that recruiting practitioners with high levels of self-efficacy can contribute to the organization on both the individual and team levels. This article also describes the research and its findings and discusses specific recommendations based on the research.

Self-efficacy is a characteristic that distinguishes between individuals according to their tendency to perceive difficult events as challenging and to what extent they feel capable of accomplishing almost any task [1]. Based on the research presented in this article, we suggest considering self-efficacy as one selection criterion for software practitioners. Recruiting practitioners with high levels of self-efficacy may contribute to the organization—not only with individuals, but also on the team and organization levels.

Our research was conducted in a large international communication company located in Israel. During Phase I, we explored for two job levels (i.e., senior versus junior) personal characteristics that can predict practitioners' orientation towards cooperative software development environments, such as Agile ones. Self-efficacy was found to be a crucial factor for practitioners in senior positions only [2]. This finding led us to focus (in Phase II) on how different levels of self-efficacy are related to software practitioners' perception of their working environment.

This article describes the research and its findings and discusses specific implications derived from the findings with respect to the recruitment processes of software practitioners. One of our primary contributions, we suggest, is the wider perspective offered on self-efficacy in the context of software organizations. Specifically, our research indicates that self-efficacy is also related to software practitioners' perception of their work environment.

Phase I: Job-Level Comparison of Agile Orientation

The first phase of the research (see [2]) examined how software practitioners' personal characteristics are related to their Agile orientation. Research participants comprised 376 software practitioners employed in two divisions of the company. In terms of job level, the sample included 228 experts and managerial-level practitioners (61 percent) and 148 junior-level practitioners (39 percent).

The research variables examined in Phase I were:

- **Agile Orientation.** Agile orientation was determined by examining practitioners' perceptions of how Agile software development takes place in practice. It was measured using eight out of the 23 items from the original version of Hazzan and Dubinsky's [3] questionnaire. The items address customer expectations, teamwork habits, and perceptions.
- **Self-efficacy.** Software practitioners' level of self-efficacy was measured using the new general self-efficacy scale, as adapted from [4].
- **Psychological Needs.** McClelland's needs theory [5] and needs survey [6] were used. The needs theory attempts to explain and predict attitude and behavior based on an individual's internal needs. Software practitioners' level of psychological needs is examined using 35 items that appeared on the original version of the needs survey, addressing the following five needs: achievement, dominance, affiliation, difference, and attitudes toward change.
- **Perceived Supervisory and Co-workers' Support.** Software practitioners' perceptions of co-workers' and supervisory support was measured using 26 of the 44 items appearing on the original questionnaire developed by Halpin and Croft [7]. This variable includes seven indicators, four of which were adopted due to their correspondence with the topic of this study (software development environments): cooperation, morale, intimacy, and manager supportiveness.

"Recruiting practitioners with high levels of self-efficacy may contribute to the organization—not only with individuals, but also on the team and organizational levels."

All research variables were rated on the Likert scale (where 1 indicates a low perceived level of the measured item and 5 indicates a high perceived level)[1].

Specifically, by using the structural equation modeling analysis method [8, 9], Phase I examined relations between Agile orientation and psychological needs, self-efficacy, and perceived supervisory and co-worker support at junior and managerial-level practitioner job levels. Main findings in this phase included:

- For both job levels, perceived supervisory and co-worker support seem to be a crucial factor that mediates the relations between the individual's different psychological needs and his or her Agile orientation. Specifically, a high level of perceived support is positively related to a high level of orientation. This finding is important since it has direct implications for the design of a software development environment.
- For managerial-level practitioners only, a high level of self-efficacy is positively associated with a high level of perceived support and a high level of Agile orientation.

Further details on Phase I and explanations of the results are presented in [2].

The fact that self-efficacy is frequently researched in organizational behavior studies—and that it appears in our Phase

I research as a variable that dominates perceived support and Agile orientation only among managerial-level practitioners—led us to examine its role in software practitioners' perceptions of their work environment. Such an examination suggests that self-efficacy may also be indicative of practitioners' perception of climate (expressed by their perceived support). In other words, along with acting as an indicator on the individual level, self-efficacy is an indicator that is related to the team and organizational levels. The following sections elaborate on self-efficacy and describe the second phase of the research, where self-efficacy played a central role.

Self-Efficacy

The concept of self-efficacy has received increased attention in organizational research over the past two decades [10]. Bandura defines self-efficacy as the "belief in one's capabilities to organize and execute the courses of action required to produce given attainments" [1]. Thus, the higher one's self-efficacy is, the more likely that he or she engages and persists in task-related behavior. Research showed that self-efficacy positively predicts job attitudes [11], training proficiency [12], and job performance [13], and acts as a buffer to improve the negative effects of work stressors on employees' psychological well-being [14].

Self-efficacy has also gained some attention in the software engineering research, where it is addressed mainly with respect to competence. For example, Paul J. Ambrose proposed that in order to obtain a holistic assessment of competence, it is essential to evaluate developer perceptions and beliefs on what they can achieve—since these beliefs can impact their performance, regardless of the skills the developer possesses [15]. For this purpose, Ambrose developed a measure of developer self-efficacy to assess a critical facet of developer competence. In [16], the self-efficacy model is used in the context of knowledge sharing. The authors concluded that a software manager (or other managers) can easily look at the inputs and outcomes of the model and see where he or she could positively affect tacit knowledge sharing. The authors of [17] investigated factors associated with software developers' intention to reuse software assets. They found that technological-level (infrastructure) and individual-level (reuse-related experience and self-efficacy) factors were major determinants of the developers' behavior.

	LOW Self-Efficacy (≤ 3.75) (N = 55)		HIGH Self-Efficacy (≥ 4.75) (N = 80)		Wilks' Lambda	Beta[4]	*F*
	Mean	**SD**	**Mean**	**SD**			
Agile Culture Perceptions	3.20	.45	3.51	.62	.93	.33	10.34**
Cooperation	3.17	.51	3.47	.62	.94	.14	8.33**
Morale	3.40	.66	3.86	.73	.91	-.07	13.84***
Intimacy	3.24	.54	3.66	.65	.90	.09	15.42***
Manager Supportiveness	3.54	.44	3.88	.69	.93	.22	10.38**
Achievement	3.89	.45	4.48	.39	.67	.37	65.21***
Dominance	3.10	.51	3.71	.55	.76	.45	42.84***
Affiliation	3.95	.34	4.36	.42	.79	-.01	35.26***
Difference	3.93	.42	4.18	.56	.95	.16	7.66**
Attitudes Toward Change	3.80	.61	4.39	.51	.78	.40	37.17**

Wilks' Lambda[5] (1,133) = .52 Eigenvalue[6] = .91 Canonical Correlation[7] = .69
Chi Square (10) = 83.02, p ≤ .001

**p < .01; ***p < .001 SD=Standard Deviation

Table 1: *Summary of Canonical Discriminant Functions and Standardized Canonical Discriminant Function Coefficients*

Phase II: Self-Efficacy in Software Practitioners' Profile

Phase II of the research focused on the role of self-efficacy in practitioners' perceptions of their work environment. Specifically, we wanted to determine whether or not it was possible to distinguish between practitioners with different (high and low) levels of self-efficacy using the variables included in our research. If it is possible, we could argue that software organizations may benefit from the recruitment of practitioners with a high level of self-efficacy, not only on the individual level (based on the general studies on self-efficacy), but also on the team/organization level (based on the results of Phase I of our research).

Discriminant function analysis[2] was used to explore the research hypothesis—that is, the ability to differentiate between software practitioners according to their level of self-efficacy by relying on the other Phase I variables. In general, this analysis method enables the determination of which variables discriminate between two or more groups. The basic idea underlying discriminant function analysis is to establish whether groups differ with respect to the mean value of the variables, and then to use these variables to predict group membership (e.g., of new cases). If the mean values of the variables are significantly different in different groups, then we can say that these variables *discriminate* between the groups.

Specifically, given two groups, the discriminant function analysis selects from the set of the research variables, a subset of variables that significantly distinguish between the two groups (significance level of at least $\mathbf{p < .05}$). In the case of a single continuous variable, a **t** test is used to determine whether or not a variable discriminates between groups; in the case of nominal variables, the chi-square ($\chi 2$) test is used and for ordinal variables, and a Mann-Whitney test[3] is employed. In such cases, the **F** value for each variable indicates its statistical significance in the discrimination between groups; that is, **F** is a

Table 2: *Classification Results for Self-Efficacy Levels*

	Predicted Level		
	LOW (≤ 3.75)	**HIGH (≥ 4.75)**	**Total**
LOW (≤ 3.75)	**47 (85.5%)**	8 (14.5%)	55 (100%)
HIGH (≥ 4.75)	11 (13.7%)	**69 (86.3%)**	80 (100%)

measure of the extent to which a variable makes a unique contribution to the prediction of group membership.

To strengthen the results of this research approach, a sub-sample of the 376 practitioners was examined. It included only those practitioners who could be clearly characterized with either low levels of self-efficacy (lower than 3.75, 55 practitioners) or high levels of self-efficacy (higher than 4.75, 80 practitioners). Figure 1 presents the means of the research variables for these practitioners.

Table 1 (see previous page) presents the **F** values of our research variables, calculated to distinguish between software practitioners with high or low levels of self-efficacy. It also shows that all research variables used in Phase I are included in this set of variables; in other words, they all contribute to the discrimination between the two groups of software practitioners. Figure 1 reflects this claim by illustrating that for each research variable, its mean for participants with low self-efficacy is lower from its means for participants with high self-efficacy. The logical conclusion from this data is that all these variables contribute to the discrimination between the two groups.

Specifically, as Figure 1 shows, practitioners with high self-efficacy tend to have a greater Agile orientation than do low self-efficacy practitioners; in addition, high self-efficacy practitioners are:

- More cooperative.
- Have a greater sense of morale working with their team members.
- Feel that their personal relationships with co-workers are closer.
- Get better managerial support.
- Report higher needs in achievement, dominance, affiliation, and difference.
- Have better attitudes towards change.

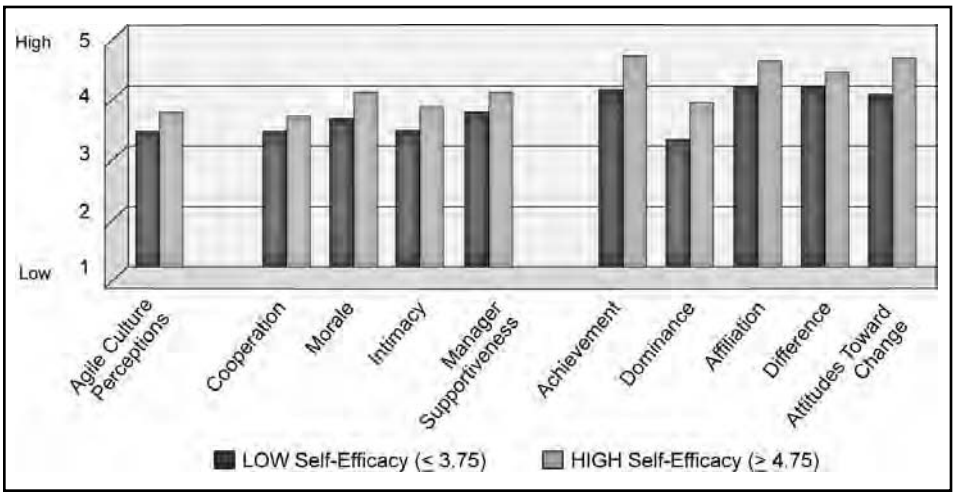
A close examination of each of these variables clearly justifies their relative value with respect to practitioners with low and high levels of self-efficacy.

Table 2 (see previous page) shows that the canonical discriminant function, constructed by the discriminant function analysis by using the other research variables, classifies correctly high-percentage of the practitioners 85.9 percent[8] according to their level of self-efficacy. Specifically, the function classifies 85.5 percent of the low self-efficacy participants and 86.3 percent of the practitioners with high level of self-efficacy. Visually, the numbers in **bold** print indicate correct classification of 47 out of the 55 practitioners with low self-efficacy and of 69 out of the 80 practitioners with a high level of self-efficacy.

"... self-efficacy can also be used as an indicator of whether or not a practitioner perceives his or her environment as being supportive—a perception that is positively correlated with development environments that encourage teamwork and cooperation."

Based on the integration of the presented results, we suggest that since the research variables both help in distinguishing between practitioners with high and low levels of self-efficacy and are relevant for software development environments, it is appropriate to use self-efficacy as an indicator for an individual's perception of his or her development environment.

Figure 1: *Means of Research Variables for Practitioners with High and Low Levels of Self-Efficacy*



Recommendations

We recommend using self-efficacy beyond its current role as an indicator of practitioners' performance. Specifically, our research indicates that self-efficacy can also be used as an indicator of whether or not a practitioner perceives his or her environment as being supportive—a perception that is positively correlated with development environments that encourage teamwork and cooperation, such as Agile software development. Based on this finding we can suggest, for example, that recruiting practitioners with high levels of self-efficacy may contribute to the organization on the individual level by recruiting practitioners with high achievements and strong beliefs in their performance, but also on the team and organizational level. Accordingly, we propose that recruiting software practitioners with high levels of self-efficacy—an easy-to-measure individual characteristic utilizing available questionnaires—may foster the formation of a supportive work climate. Needless to say that when making such decisions, each company should check very carefully its full set of considerations and characteristics and decide on the importance attributed to each factor in its recruitment processes.◆

References

1. Bandura, Albert. *Self Efficacy: The Exercise of Control*. New York: W.H. Freeman, 1997.
2. Seger, Tali, Orit Hazzan, and Ronen Bar-Nahor. *Agile Orientation and Psychological Needs, Self-Efficacy, and Perceived Support: A Two Job-Level Comparison*. Proc. of the Agile 2008 Conference. Toronto, Canada. 4-8 Aug. 2008: 3-14.
3. Hazzan, Orit, and Yael Dubinsky. *Clashes between culture and software development methods: The case of the Israeli hi-tech industry and Extreme Programming*. Proc. of the Agile 2005 Conference. Denver. 24-29 July 2005: 59-69.
4. Chen, Gilad, Stanley M. Gully, and Dov Eden. "General self-efficacy and self-esteem: toward theoretical and empirical distinction between correlated self-evaluations." *Journal of Organizational Behavior* 25.3 (2004): 375-395.
5. McClelland, David C. 1961. *The*

Achieving Society. Princeton, N.J.: D. Van Nostrand, 1961.
6. McClelland, David C. *Human Motivation*. New York: Cambridge University Press, 1987.
7. Halpin, Andrew W., and Don B. Croft. 1963. *The Organizational Climate of Schools*. Chicago: Midwest Administration Center of the University of Chicago, 1963.
8. Kline, Rex B. *Principles and Practice of Structural Equation Modeling*. New York: The Guilford Press, 1998.
9. Mueller, Ralph O. *Basic Principles of Structural Equation Modeling*. New York: Springer-Verlag, 1996.
10. Chen, Gilad, and Paul D. Bliese. "The Role of Different Levels of Leadership in Predicting Self and Collective Efficacy: Evidence for Discontinuity." *Journal of Applied Psychology* 87.3 (2002): 549-556.
11. Saks, Alan M. "Longitudinal field investigation of the moderating and mediating effects of self-efficacy on the relationship between training and newcomer adjustment." *Journal of Applied Psychology* 80.2 (1995): 211-225.
12. Martocchio, Joseph J., and Timothy A. Judge. "Relationship between conscientiousness and learning in employee training: mediating influences of self-deception and self-efficacy." *Journal of Applied Psychology* 82.5 (1997): 764-73.
13. Stajkovic, Alexander D., and Fred Luthans. "Self-efficacy and work-related performance: A meta-analysis." *Psychological Bulletin* 124.2 (1998): 240-261.
14. Jex, Steve M., and Paul D. Bliese. "Efficacy beliefs as a moderator of the impact of work-related stressors: A multi-level study." *Journal of Applied Psychology* 84 (1999): 349-361.
15. Ambrose, Paul J. "Metacognition and software developer competency: construct development and empirical validation." *Issues in Information Systems* 6.2 (2003): 273-279.
16. Endres, Megan L., et al. "Tacit knowledge sharing, self-efficacy theory, and application to the Open Source community." *Journal of Knowledge Management* 11.3 (2007): 92-103.
17. Mellarkod, Vidhya, et al. "A multi-level analysis of factors affecting software developers' intention to reuse software assets: An empirical investigation." *Information & Management* 44.7 (2007): 613-625.

Software Defense Application

There has been the long-held idea that individual success increases from self-efficacy: one's belief in their ability to complete a project. Success in any project and in any organization—including in the DoD—can be buoyed by high levels of self-efficacy. Though one can argue that self-efficacy by itself is sufficient for the creation of a successful team, our findings emphasize that the way in which the environment is perceived by individuals should be considered as well. Through understanding these different perceptions, and recruiting practitioners who have high levels of self-efficacy, individuals, teams, organizations, and their products, all improve.

Notes

1. In addition, two background variables—years of experience and age—were collected; they were not, however, included in this analysis.
2. Our description of the discriminant function analysis is partially based on <www.statsoft.com/textbook/discriminant-function-analysis>.
3. For more on Mann–Whitney and chi-square testing, see <http://en.wikipedia.org/wiki/Mann-whitney> and <http://en.wikipedia.org/wiki/Chi-square_test>, respectively.
4. The betas are the coefficients of the unstandardized discriminant function. Each subject's discriminant score is computed by entering his or her variable values (raw data) for each of the variables in the equation. The betas are used to construct the actual prediction equation, which can then be used to classify new cases. In our case, see Table 2.
5. Wilks' Lambda is the proportion of total variance in the discriminant scores not explained by differences among groups. A lambda of 1.00 occurs when observed group means are equal. A small lambda indicates that group means appear to differ. Here, the Lambda of 0.52 has a significant value (Sig. < 0.001); thus, the group means appear to differ.
6. Eigenvalue: A large eigenvalue, as present in our case, indicates a high proportion of explained variance in the predicted variable (in our case, level of self-efficacy).
7. The canonical correlation is the multiple correlation between the discriminant scores and the levels of the dependent variable. A high correlation indicates a function that discriminates well. The present correlation of 0.69 is not very high (1.00 is perfect).
8. Determined by adding the number of employees at high and low levels (47+69) divided by total practitioners (55+80).

About the Authors

Orit Hazzan, Ph.D., is an associate professor at the Department of Education in Technology and Science of the Technion – Israel Institute of Technology. In 2004, she co-authored "Human Aspects of Software Engineering" with the late Jim Tomayko. Her second book (co-authored with Yael Dubinsky), "Agile Software Engineering," was published in 2008. Hazzan is a consultant for several software projects in the Israeli software industry.

Technion
Israel Institute of Technology
Department of Education in Technology and Science
Haifa 32000
Israel
E-mail: oritha@tx.technion.ac.il

Tali Seger, Ph.D., is a senior lecturer in the Ruppin Academic Center and the head of Human Resource and Organizational Development in their MBA program. From 1998-2004, Seger was an organizational consultant and worked as a human resources manager at several Israeli companies. She received her doctorate in business administration from Haifa University in 2006.

Ruppin Academic Center
Department of Business Administration
Emek Hefer 40250
Israel
E-mail: talis@ruppin.ac.il

From Projects to People: Shifting the Software Acquisition Paradigm

Dr. Douglas J. Buettner
The Aerospace Corporation

Lt. Col. Chad Millette
United States Air Force

The amount of embedded flight software is growing at a tremendous rate in the National Security Space (NSS) systems under development by the Space and Missile Systems Center (SMC). Problems with Total System Performance Responsibility (TSPR)-era programs like the Space-Based Infrared System (SBIRS) have been aligned with opinions that the DoD has lost the "recipe" for acquiring complex space systems. The software-intensive nature of next-generation space systems necessitates consideration of a new software-intensive system acquisition paradigm to not only take full advantage of the best people that defense contractors have to offer, but to ensure the ability to engineer and build these systems far into the future.

In October 2006, Lt. Gen. Michael Hamel [1], the SMC's Program Executive Officer, briefed the SMC system software growth trend to the National Defense Industry Association Defense Software Strategy Summit (see Figure 1).

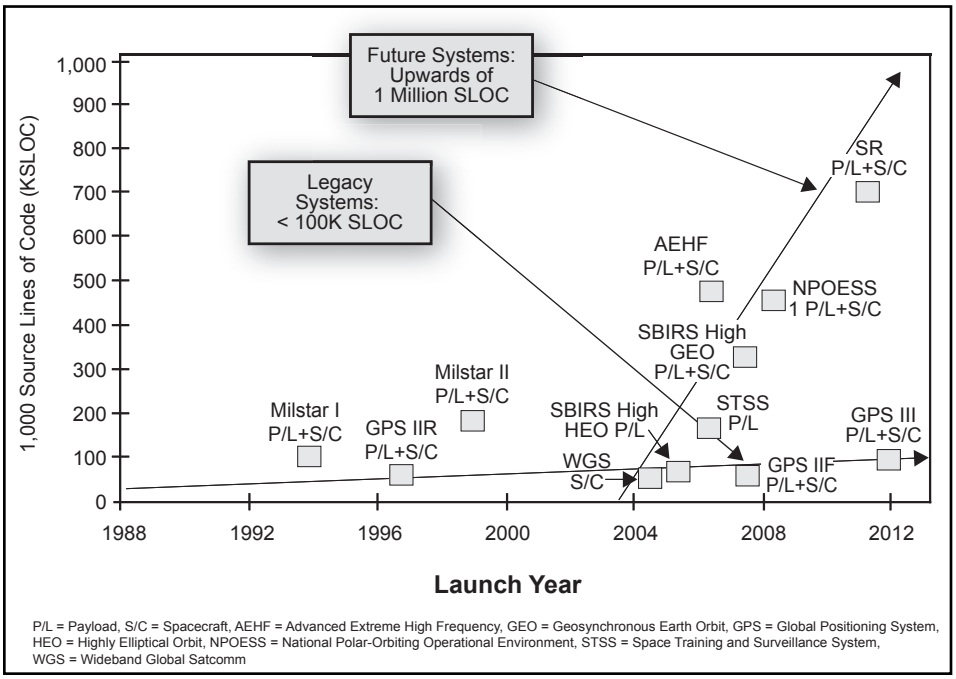
In [2], Buettner and Arnheim described the SMC-wide review of test issues attributed to the TSPR-era acquisition reform policy changes and its impacts on embedded flight software; also provided were space computer technology improvement reasons for the observed growth trend. While the legacy class of vehicles (shown in Figure 1) appear to have a manageable growth trend, the software growth trend for the future space systems (with envisioned systems greater than one million SLOC) is beyond anything our space defense establishment has had to grapple with in the past.

Hamel's presentation supported a broad industry software strategy summit report [3] containing the following recommendations (among others):

- Review and analyze the software engineering, acquisition, and life cycle management initiatives, policies, processes, and plans. This should occur in service branches (Army, Navy, and Air Force), defense agencies, and in other organizations such as the National Security Agency.
- Solicit service branch, major command, engineering center, and Program Executive Office software life-cycle management recommendations.
- Define and publish the DoD's long-term objectives and course of action with associated priorities and resources in a software life-cycle strategy.

In the face of increased software demand, software project difficulties, limited experienced personnel availability, varied standards and processes, and declining budgets, the report recommends that DoD management staff continue aggressively focusing on "software engineering, acquisition, management, and human resource life-cycle challenges through the application of resources and focused action" [3].

Figure 1: *Software Growth Trend* [1]



Fundamentally, many of the problems are a side effect of the DoD's current competitive bid acquisition strategy. It is our belief that a number of the problems could be minimized using a paradigm shift away from competing for the software engineering and development aspect of these software-intensive contracts. Hence, we provide supporting arguments and information showing that a number of the issues that we have faced on the SBIRS—and those facing other software-intensive system acquisitions—are side-effects attributable to constraints imposed by the competitive-bid acquisition process. These constraints stress cost and schedule from the onset, resulting in additional rework cycles from the late discovery of quality issues. Furthermore, we will explain how a paradigm shift could minimize these issues for the class of space system acquisitions that are on the future systems software growth trend. The current acquisition paradigm involves a competitive bid (with software as a factor) between teams of contractors in response to a request for proposal (RFP).

The Problem With "Best Value" Bidding

In a Defense Acquisition University (DAU) course class exercise (attended by Millette), students assumed the roles of contractors preparing a bid response to an RFP for a software-intensive system. Students are given three options for software costs: a low-, medium-, and high-cost figure. The evaluation criteria indicated that cost was not specifically a criterion, but it is certainly always considered.

Having the development life-cycle

issues faced by the SBIRS in their background, the group selected the high software cost as a way to mitigate the overall software development risk. The students believed that the high-cost figure would, combined with staying under the life-cycle and unit procurement cost thresholds: reduce overall cost and schedule risk; help inevitable requirements creep, rework, and other typical software cost and schedule drivers; and present a better risk-mitigation position. However, when the other groups briefed their analysis of the proposal—and specifically, why they did not recommend selection—each cited the high software cost as a negative aspect of the proposal.

In the SMC, we learn this lesson often: It is not in the contractor's best interest to bid the actual, expected, or risk-sensitive cost, as the evaluators may not recognize this as a positive and will only focus on the bottom line. The contractors we work with are not devious or intentionally trying to underbid these efforts maliciously; they are simply doing what they believe they have to do in order to secure the work. If one bidder of the group goes with the realistic or conservative cost estimate, they run the risk of being identified as not providing the best value bid for the government.

From this experience—and some of the observed strategies employed by the bidders for the SBIRS program and others—we conclude that contractors will (and do) try to utilize cost-minimization strategies to win contracts. If they are bidding on multi-billion dollar systems, cutting costs to save the government billions of dollars has repeatedly been shown to be a winning strategy. While shaving costs in an attempt to provide the government and the taxpayer with a system for a good value is appreciated, it ultimately raises the question of how such strategies could impact the quality of the NSS mission's critical software component early in a program's life cycle.

In [4], seven different flight software projects contained in an Aerospace Corporation database are reviewed: Two remarkably different software projects are compared in detail using a system dynamics model. Chapters focused on qualitative research and game theory provide a number of insightful government schedule and cost pressure strategy impacts on contractor quality. Also included is a model showing the deleterious schedule impacts from the early life-cycle schedule-driven behavior: minimizing effort in quality-enhancing peer reviews and developer unit testing (that is often perceived by these individuals to be a waste of time). Contrary to the near-term schedule-saving efforts by engineers, the opposite schedule effect occurs in the long-run due to the increased time spent fixing errors that are found later (e.g., during software integration testing).

The application of game theory concepts (see sidebar) suggests how contracts can get into the situation that TSPR policies seemed to accentuate. TSPR policies both reduced government oversight leading to contractor decisions contrary to government's quality expectations, and removed software development standards from the contracts. Software standards are essential on contracts: They result in development and test practice compliance that all contractors use to achieve a rigorously engineered software product with a demonstrated level of quality required by space systems. Thus, when it comes to software quality, strategies for bidding low will inevitably lead to cost and schedule overruns.

Reference [4] provides 26 specific recommendations for the government and contractors, including controversial subjects like mandating certifications for software professionals. However, as long as we continue to competitively bid software (as an integral part of NSS systems), the cost and schedule driven aspects faced by the SBIRS program will persist—if not get even worse—in the future. It is our belief that these issues are founded in the government's competitive bid approach, therefore making the current acquisition model unsustainable—even using newer model-based software development methods utilized by the automotive industry pushing cars into the 10 million and 100 million LOC regime. The adoption of these development methodologies into embedded space systems will undoubtedly help; however, due to the nature of the bidding process for these unique and extremely costly systems, we contend that they do not address the fundamental issues[5].

Game Theory and the Bidder's Billion-Dollar Dilemma

Game theory can be used to analyze optimal strategies for action in competitive situations with two or more players of the game[1]. Game theorists use a strategy matrix to analyze each player's strategies when they attempt to take into account the action of their opponent in their decision-making process in order to maximize their payoff[2]. An example of a normal form game matrix with two distinct strategies (*A* and *B*) has Player 1 payoffs of α_1 for the *A* strategy and β_1 for the *B* strategy, while Player 2 payoffs are α_2 for the *A* strategy and β_2 for the *B* strategy:

		Player 2	
		A	*B*
Player 1	*A*	(α_1, α_2)	(α_1, β_2)
	B	(β_1, α_2)	(β_1, β_2)

The game is called a zero-sum game when one player's payoff (win) is the other player's loss. However, the game is called a non-zero sum game if the payoff to the winning player is not from the losing player[3].

We now consider a case where an acquisition game for a very expensive billion-dollar satellite system results in only two potential bidders, with the government acquirer responsible for setting up the rules of the game. Both bidders have two pure strategies: the *A* strategy results in providing a bid for non-recurring research and engineering to build the system that incorporates more costly risk mitigation techniques leading to a politically unacceptable cost plus a substantial fee if they win; and a *B* strategy that results in an acceptable cost plus a substantial fee if they win. If they lose, they simply get reimbursed for their effort to create a bid. Mathematically we write:

$$(\alpha_1 = C + f_1) >> (c + f_1 = \beta_1) \text{ likewise}$$
$$(\alpha_2 = C + f_2) >> (c + f_2 = \beta_2).$$

In this situation, the *A* strategy with its higher cost risk mitigation activities is considered a losing strategy. The highly desired *B* strategy solution (in this case) is a Nash equilibrium[4]. Unless both bidders were required to include the cost for those risk mitigation activities in their bids, the likely outcome is bids that removed these engineering tasks.

In addition to the TSPR policy changes that directly impacted space system software development and testing standards previously mentioned, the cost as an independent variable (CAIV) strategy was envisioned as a method for government to control cost by making it a constraint [14]. Consider the impact of these constraints at the software engineer level: Tough cost, schedule, and quality tradeoff decisions need to be made when trying to hire the people required to complete the contractually obligated design documents, write the software code, and test the software. In addition, staffing is required to adequately review the design, code, and test products. Experience has often shown that sound engineering judgment becomes dominated by what is believed to be *good enough*.

Hence, we propose an alternative software acquisition paradigm that we believe can work to effectively minimize a number of these issues: Remove the competition for low-cost from consideration for the software component of the system acquisition.

Causes of Project Success—and Failure

Ivar Jacobsen, Grady Booch, and James Rumbaugh identify software success factors as dependent on people, process, product, project, and tools [15]; it can also be argued that process, product, project, and tools are also fundamentally dependent on people, and thus people are central to the entire software problem. While establishing an early version of COCOMO, Barry Boehm found that success—in regards to lower costs and on-schedule delivery—was highly dependent on the software team [16]. Considering that contractors are forced to manage and change out the personnel they hire and retain to build our systems (based on the contracts they are able to win), we are faced with a situation where we are dynamically dependent on the number and quality of personnel available at any given time. Even true A-teams under adverse cost and schedule constraints have a high probability of significantly overrunning cost and schedule in order to maintain quality. However, projects driven by cost are not likely to get the best people. The result is a mixed bag of really good people trying to pull along a cadre of less-capable performers that help bring staffing numbers up to the appropriate number the cost/schedule models suggest. These models allow for dialing in the capability of teams; our experience from a number of programs, however, shows that contractors always claim that they have the A-team, that they are CMMI® Level (fill in your favorite contractually obligated number here), and that their team can write the software faster than the speed of light with virtually no defects. When the project is finally over (after either numerous cost overruns or finally being cancelled), these people are recycled onto the next project. Hence, people—more specifically, the assignment, organization, and overall management of people—are the Achilles' heel of software.

"... a properly managed labor pool could provide cost advantages as well as a method for identifying and retaining the best engineering talent."

Oftentimes, projects are saddled with the following problems, all leading to late life-cycle schedule and cost overruns:

- Early life-cycle personnel lack the fundamental knowledge required to successfully write requirements or to design, build, and mathematically test complex real-time satellite control software.
- Management does not appreciate the need for following documented engineering processes.
- Cost- and schedule-driven decisions, mandated by government action, remove numerous prudent risk-mitigation steps.

Once upon a time, we could hide our software foibles behind extremely visible hardware issues, but not any longer.

Establishing a National Systems Engineering Laboratory

Quality and schedule could be met (at least within a consistent cost and schedule margin) if we fundamentally shift our acquisition paradigm: from program-by-program cost and schedule management to a focus on the quality of people used to feed our engineering teams. This would be accomplished by establishing what we call a National Systems Engineering Laboratory (NSEL). In it, the private sector (the big system integration and Systems Engineering and Technical Assistance contractors) are initially reimbursed to provide these facilities with their very best software personnel (management and engineers). The NSEL would also be cooperatively staffed with selected personnel from our universities, federally funded research and development centers (FFRDCs), and government services.

As new systems are being competitively designed by select senior private sector staff (competing the hardware and model-based software designs based on system requirements), they are supplemented by NSEL personnel and high-performing university students working behind strict firewalls. NSEL and student personnel would have the experience and training to provide an initial set of documentation and prototypes to acquisition staff. The winning contractors, supplemented with these NSEL (and now more mature) student personnel, would build, from the preliminary design and prototype, the final software.

Standard systems and software development process tools—such as the Agile, Spiral, or incremental models—are used insomuch as they are tailored by the engineers themselves to follow the best practices brought forth from industry and academia, based on each system's size and type of effort. Personnel are trained and incentivized to both follow these processes and also suggest process improvements as lessons are learned and technology advances. Overtime policy can be set to maintain schedule, but never to the detriment of quality. Incentives are provided to ensure creation of only the end-products necessary for designing the system and the documents that must be handed off to the next development phase or as required to maintain the system.

Prestige combined with attractive pay and high quality of life, NSEL site locations can be used to attract the best of the best. Contractor payouts are used to entice industry to bring to the table for consideration their best processes, software designs, and existing code used in other systems. Once the final system design has been selected, the pool of available top-notch engineers can draw from a wealth of software designs and prototype code to build the final flight code. Existing systems in use can draw from the same pool of engineers to maintain these systems, as needed.

Another consideration is utilizing university students and fresh graduates as a significant labor source. Software-inten-

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

sive systems are usually dependent on the development and maintenance of significant specialized software utilities and tools to support the development effort. Select students using an open-source software paradigm could interact with the top-notch NSEL engineers (as their customers) to develop the required tool suites. While open-source code in our defense systems is usually fraught with security concerns, a properly managed labor pool could provide cost advantages as well as a method for identifying and retaining the best engineering talent. Expanding this open-source tool suite support effort to include actual system code could be investigated once the paradigm takes hold.

This alternative paradigm would alleviate the dilemmas facing prospective bidders on software-intensive acquisition efforts (exemplified in the DAU example). Under the NSEL paradigm, underbidding the software development cost would be unnecessary because it would not be a direct labor charge to the contractor. The late life-cycle software development personnel would be supplied by the NSEL, and could tap into prototypes from our universities, the contractor community, and the engineered design for the target system. This would ultimately lead to more predictability in the cost and schedule of the software development efforts, as the NSEL would employ high-quality people using disciplined processes tailored for the specific acquisition underway.

The paradigm—that funds an NSEL as a national asset, and removes the software cost consideration from the bidding process—includes the following:

- There will be competition between engineering teams.
- Early software activities will provide risk mitigation for the construction phase of the contract.
- Independent government and support staff will evaluate the engineering designs and estimated construction costs for the different systems.
- NSEL staff will be included on each team, with the expectation of working in seclusion from NSEL members on the other team.
- Teams will individually utilize the pool of available sub-contractors—with products and services determined by the systems and software engineering each team requires to build the system.
- The engineering and software prototyping staff is selected based on merit, capability, and need.
- Software build and test staff is selected in part from the NSEL staff on the losing team and from engineering/software prototyping staff. They will test the constructed software system or augment the software development and test phase, based on staffing requirements.

The predicted end-result is a higher quality software product that is staffed with the best people available. However, it should be mentioned that the number of new software-intensive systems we could build as a nation would be constrained by the number of NSEL staff. Yet, we view this as an acceptable engineering alternative to the CAIV-approved approach under TSPR.

The NSEL is first and foremost tasked with building and maintaining quality systems, with a strong eye towards successfully designing and building cost- and schedule-acceptable solutions. Under this premise, quality staff can, with time, create their own training and competitive hiring policies for their engineering positions. In this manner, the processes developed and promulgated by these staff tend (through generations of engineers) towards a level that can keep up with demand. While problems will undoubtedly arise, this self-contained, continual learning environment will foster and lead towards solutions for these issues.

We also propose that NSEL directors for functional areas in engineering are hand-selected for prestigious appointments from academia, the FFRDCs, and private industry. Their hiring will be based on current requirements for government positions—and will not be hired via political appointment. The directors' primary role will be to resolve technical and management issues—with the national need, which is always at the center of their decision-making process.

Conclusions

An NSEL for defense acquisition strategy is an alternative system acquisition concept that is based on a grass-roots or grounds-up negotiation between the engineering disciplines. It has the potential to take the DoD boldly where no one has gone before—allowing for acquisition, next-generation-embedded, software-intensive systems. This grass-roots process is designed to provide the best quality-minded engineers needed to yield engineered systems with a consistent cost and schedule. We also believe that this concept is required to mitigate the software-intensive, system-driven people factors that have plagued a number of our system acquisitions—leading some to believe that we have lost our space acquisition recipe for success. We acknowledge that the concept must pass through the normal gamut of politically driven negotiations. Hopefully, during this process, the concept for building a national capability consisting of the best of the best—and a method to identify and retain our university engineering talent—is not lost. We've even heard of an even more drastic strategy: using a draft to nationalize our software development and system engineering talent. Short of this controversial and unlikely option, creating a prestigious system engineering research and development laboratory—retaining and nurturing the world's best engineering talent—is a sound method, fundamentally based on the talent-retention successes of our national laboratories. Our national goal should be to attract the best personnel to this field and we suggest that subtleties such as funding details, redundant locations, and other issues can all be politically negotiated and worked as this concept is matured.◆

References

1. Hamel, Michael. *Growth Trend in System Software*. Proc. of the Defense Software Strategy Summit. Arlington, VA. 18-19 Oct. 2006 <www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Content/ContentGroups/Divisions1/Systems_Engineering/Hamel_10_18.pdf>.
2. Buettner, Douglas J., and Bruce L. Arnheim. *The Need for Advanced Space Software Development Technologies*. Proc. of the 23rd Aerospace Testing Seminar. Manhattan Beach, CA: 10-12 Oct. 2006.
3. Office of the Deputy Under Secretary of Defense for Acquisition and Technology Systems and Software Engineering. *Defense Software Summit Report*. Final Draft, Vers. 1.0. Washington D.C.: Pentagon. Oct. 2006 <www.acq.osd.mil/sse/docs/NDIA-Defense-Software-Summit-Report-October-2006.pdf>.
4. Buettner, Douglas J. "Designing an Optimal Software-Intensive System Acquisition: A Game Theoretic Approach." Ph.D. Dissertation. Astronautics and Space Technology Division, Viterbi School of Engineering, University of Southern California, Sept. 2008.
5. Schinasi, Katherine V. "Military Space Operations: Common Problems and Their Effects on Satellite and Related

Software Defense Application

The article proposes a National Systems Engineering Laboratory that can help other DoD entities with issues that the authors have observed in their National Security Space software-intensive system acquisitions. This article first details the problem of cost mitigation bidding strategies used by DoD contractors, and then recommends solutions through an NSEL. Removing this competitive bid process for the software engineering through the creation of a prestigious systems engineering lab—staffed by our nation's best engineers, created solely to provide quality design and engineering services—is one possible solution.

Acquisitions." Letter to The Honorable Jerry Lewis. U.S. General Accounting Office. GAO-03-825R Satellite Acquisition Programs. 2 June 2003 <www.gao.gov/new.items/d03825r.pdf>.

6. GAO. *Defense Acquisitions: Improvements Needed in Space Systems Acquisition Management Policy*. GAO-03-1073. 15 Sept. 2003 <www.gao.gov/new.items/d031073.pdf>.
7. GAO. *Defense Acquisitions: Despite Restructuring, SBIRS High Program Remains at Risk of Cost and Schedule Overruns*. GAO-04-48. 31 Oct. 2003 <www.gao.gov/new.items/d0448.pdf>.
8. GAO. *Space Acquisitions: Stronger Development Practices and Investment Planning Needed to Address Continuing Problems*. GAO-05-891T. 12 July 2005 <www.gao.gov/new.items/d05891t.pdf>.
9. GAO. *Space Acquisitions: DoD's Goals for Resolving Space-Based Infrared System Software Problems Are Ambitious*. GAO-08-1073. 30 Sept. 2008 <www.gao.gov/new.items/d081073.pdf>.
10. Singer, Jeremy. "SBIRS Report to Include Update on Health of Defense Support Program." *Space News*. 8 Dec. 2004.
11. Lockheed Martin. *First SBIRS Satellite With New Flight Software Completes Key Test at Lockheed Martin*. 13 Jan. 2009 <www.lockheedmartin.com/news/press_releases/2009/11309ss_sbirs.html>.
12. Defense Industry Daily. "Despite Problems, SBIRS High Moves Ahead." 12 Sept. 2009 <www.defenseindustrydaily.com/Despite-Problems-SBIRS-High-Moves-Ahead-With-3rd-Satellite-Award-05467/#more-5467>.
13. GAO – Applied Research and Methods. *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Capital Program Costs*. GAO-09-3SP. Mar. 2009 <www.gao.gov/new.items/d093sp.pdf>.
14. Perry, William J. "Acquisition Reform." *Annual Report to the President and the Congress*. Chapter 14. Mar. 1996 <www.dod.mil/execsec/adr96/chapt_14.html>.
15. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Reading, MA: Addison-Wesley, 1999.
16. Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

Notes

1. See James O. Berger's *Statistical Decision Theory and Bayesian Analysis* (1980 edition, page 310).
2. See Harold W. Kuhn's *Lectures on the Theory of Games* (2003 edition, pages 5-6).
3. See Philip D. Straffin's *Game Theory and Strategy* (1993 edition, pages 3-6).
4. See Straffin (pages 65-68) and Roger B. Myerson's *Game Theory: Analysis of Conflict* (1997 edition, pages 97-98).
5. We do not believe that current methods will solve the well-documented software development issues that have plagued government acquisition. These issues are documented in the popular press and in numerous GAO reports; see [5, 6, 7, 8, 9, 10, 11, and 12]. Furthermore, a recent GAO cost estimation and assessment guide documenting the best practices for developing and managing capital program costs singled out the SBIRS as a case study [13].

About the Authors

Douglas J. Buettner, Ph.D., is the systems director of the Flight Software Department in the Aerospace Corporation's Space-Based Surveillance Division. Buettner has more than 20 years of experience including contracts from NASA's comet sample return mission, all phases of defense industry software development, and quality assurance management for venture capital-backed startup companies. He has bachelor's and master's degrees in physics from Oregon State University and, as an aerospace fellow, completed his doctorate in astronautical engineering (with a focus on computer science) from the University of Southern California.

Phone: (310) 336-5658
E-mail: douglas.j.buettner@aero.org

Lt. Col. Chad Millette is the Deputy, Operational Requirements Branch, HQ AFMC/A5C. His previous assignment in the SBIRS Wing included responsibilities overseeing the development of the embedded flight software for the SBIRS program. He has a bachelor's degree in mathematics from the University of Arizona, a master's degree in software engineering administration from Central Michigan University, and a master's degree in systems management from the Air Force Institute of Technology. He is also a certified Project Management Professional and a Software Development Certified Professional.

4375 Chidlaw RD
Wright-Patterson AFB, OH 45433
Phone: (937) 656-3961
E-mail: chad.millette@wpafb.af.mil

Allocating Resources in Multi-Project Programs: Lessons Learned from the Trenches

Edward Lari, Dr. Jeffrey Beach, Dr. Thomas A. Mazzuchi, and Dr. Shahram Sarkani
George Washington University

There have been significant strides in the use of project management methods for resource allocation in single-project programs in the DoD environment. Unfortunately, when it comes multi-project programs, inter-project resource allocation techniques are inadequate. The objective of this article is to shed light on some of the challenges program managers are confronted with when allocating resources among multiple project programs, and to highlight some of the practical solutions that have been proven useful in resolving these issues.

The past two decades have seen an increase in the size and complexity of DoD software implementation programs. This has been accompanied by an exponential growth in the number of multi-project programs in program managers' portfolios. Yet the literature on program management has tended to be written with the underlying assumption that the programs managed are made up of single projects as opposed to multi-project programs. In reality, this is not the case. Most programs involve the simultaneous management of multiple projects or a portfolio of projects managed to ensure the aggregate results at the end of the program.

Multi-project programs further add to the level of complexity of the resource management equation. In managing large-scale software implementations, the success of any program is largely dependent on the effective utilization of its resources across multiple projects. Right from the initial stages of program initiation to the final stages of delivery, the program manager is often faced with the complexities of resource allocation and its negative impact on performance results.

The Traditional Process for Resource Allocation

In multiple-project programs, demand for project resources typically exceeds its supply. This is usually the first lesson learned by the fledgling manager involved in any aspect of the resource allocation process. This has given rise to one of the basic principles of project portfolio management: By defining the relative priority of projects and by applying resource leveling and skill-set capacity constraints, a method which allocates resources in a manner that optimizes their enterprise utilization can be derived. While this theory has many proponents—and the principle may, in fact, be true—its practical implementation is usually very challenging.

A review of traditional methods of resource allocation would illustrate how it falls short in determining the true enterprise demand or in identifying the key resource conflicts at the program level. From a conceptual standpoint, there is a great deal of commonality in the processes utilized by organizations to allocate resources across multiple projects. The tools used for this purpose, be it customized software, spreadsheets, tables, or even roundtable discussions, are all different. Yet all traditionally begin with an algorithm for assessing and categorizing demand, and creating the proposed book of work. Considering that demand will exceed supply—and that the pool of resources will run dry long before the list of initiatives are exhausted—the initial step is to give relative rankings for the different projects through a prioritization process. The prioritization process is followed by building the master project schedule, capacity planning, and high-level resource assignments.

The prioritization process doesn't require complexity, needing only an estimate of expected benefits and initial costs to be effective. For example, one small division of an intelligence organization used the very simplistic yet effective *Is it* process, which sought answers for questions such as *Is it a project?*, *Is it a realistic estimate of benefits?*, and *Is it a valid approach to solving a problem?*. Also used are resource estimate descriptions such as *Small*, *Medium*, and *Large* to facilitate the definition of a project's cost/benefit. These resource estimates—which would inevitably be tied to a project's initial priority—were developed after a 15-minute roundtable review of the proposed solutions by technology platform managers. Whatever the method is, the net results of a prioritization process are either an ordinal ranking of projects or a grouping of projects into *Critical*, *High*, *Medium*, and *Low* categories.

Once the initial prioritized list of projects is created, the next step is to build the preliminary enterprise master program schedule. The master schedule establishes time boundaries for projects that are ready to go and need to be staffed. It may also create placeholders for those projects that should remain in the queue until the next round of prioritization occurs. Projects that do not make the cut are sent to the *project graveyard* where they wither away from lack of attention or are resurrected for another crack at prioritization. Within all project portfolio management systems is an embedded project evaluation process used to assess the health of the projects at various stages in their life cycle. It is critical to ask whether the project is still relevant to the program's goals and objectives. If the answer is no, then the project should be stopped. This way, the program manager ensures that all resources are deployed where they will offer the best return to the program as a whole. In practice, this is one of the most difficult decisions program managers have to make, as reshuffling a project portfolio will likely displease the managers of those projects. However, the program manager must always seek to globally optimize the program performance—as opposed to trying to optimize each project.

After the master program schedule is created, the next step is to undertake capacity planning. Capacity planning, normally performed by the platform and skill set, is what makes the master schedule believable. Here, the total resource hours are deducted from the overall capacity. Project managers are given their projects and—with the staff they believe is required to complete those projects—go through their initiation processes with management's final instructions regarding the triple constraints: time, budget, and quality.

High-level resource assignment follows capacity planning. During the resource assignment stage, the project managers provide resources to the scheduling groups. While doing this, the possibilities of over-allocation and under-allocation arise. The result of a specific resource shortfall means that the team will be asked to do more to either cover the deficit or be forced to get by with less.

High-level resource assignments are analogous to a hotel reservation. In software engineering, the department or platform (e.g., Web development, AS/400, testing) supplying the resource is the hotel. The room and room type correspond to position (i.e., developer) and attributes (e.g., manager, senior developer, etc.) of the resource. Length of stay (duration of assignment) is allocated from available inventory, but the actual room number (resource name) is not yet known. It will not be determined until the guest (project manager) arrives (kicks off their project). The difference between the two comparisons is that in the field of IT, the project manager can't be upgraded to a better room or compensated with a free meal for any inconvenience. As mentioned, specific resource shortfalls mean that the team will be asked to do more to cover the deficit or be forced to get by with less; hotel guests, however, will never be asked to share a room with others if the hotel over-books.

Challenges With the Traditional Approach: A Case Study

So if the resource allocation process—with stages such as prioritization, scheduling, capacity planning, and high-level resource assignment—takes place under constant scrutiny and monitoring, then why do so many programs fail? And why do most analyses of program and project failures[1] list poor resource scheduling and planning as primary causes?

To plainly illustrate this point, we present the following scenario from a case study encountered by the authors in consulting assignments.

A software requirements analyst (SRA) was told that for the next five weeks their time will be spent on a project. Assuming a 40-hour work-week, this equates to an initial resource request of 200 hours. At the project kickoff meeting, the SRA is given a high-level schedule that mapped out the standard project management life-cycle milestones. According to the project manager, the SRA's role on the project would be limited to three weeks developing the functional requirements and two weeks assisting in the creation of the system design document. The project manager assured the analyst that none of these issues would take more than 20 hours a week during that time period. The analyst examined the project schedule and reported back that the SRA could make those dates. The SRA then needed to meet with three different business units to solicit their input into the document: This required first arranging times to meet with the units individually, then finding a location for a comprehensive meeting, and then using the knowledge to prepare documentation on the existing process. All of those activities can be done in a week. The analyst estimated two weeks for creating the functional requirements document, including time for walkthroughs (that need to be scheduled), revisions, and signoffs seemed reasonable. As for the time required for assisting system design, the analyst wasn't sure—but two weeks seemed realistic.

Two days into the project, the SRA realized the task estimates were off significantly. There was no existing documentation for the system: That meant two weeks of sitting down with developers and creating it. Scheduling the meetings turned into a nightmare as well. There actually turned out to be 10 business units that were stakeholders in the process—and located throughout the country. Including travel time (even meeting face-to-face with half the units and speaking to the others via conference call), this made crafting an initial draft of the requirements much more than a two-week job: four weeks if everything went well, six weeks with coordination issues. To make matters worse, one of the developers came up with a new idea on how to implement the functionality. Even though the SRA had to wait until the business requirements were complete to make sure, the developer confidently projected that his time would drop from 12 weeks to three. The project manager thought any lost requirement time could be made up during the development, and continued to report the project's overall status as *green* (on schedule within an acceptable variance of planned targets). Needless to say, the project did not move on as per the previously planned project schedule. This example clearly shows that in spite of thorough planning, something had gone wrong somewhere.

An analysis of this case study presents the following questions:

- What numbers should be reflected in capacity planning and the enterprise project schedule?
- While the aggregate hours for this project may decrease, what will be the impact to the SRA's pool of resources for the program?
- Where are the extra analysis hours supposed to come from?

The theme echoed in this scenario—one repeatedly encountered by project and program managers—is that the root cause of failing to perform the resource allocation process properly can be traced back to two specific issues:

1. The inability of program managers and the systems that support them to determine true capacity.
2. The failure of project and program managers to comprehend the impact of the program's critical path across multiple projects.

Alternative Solution Approaches

While these problems are solvable, the solutions, like most management problems, are neither linear nor discrete. Experience indicates, however, that a combination of approaches will go a long way in solving the problem.

Determining True Capacity

A common error often made by programs is failing to perform true program capacity in the initial stages. For program capacity planning to work, both the supply of and demand for available resources must be well-estimated. What may not be apparent in performing this exercise is how to account for the margin of error inherent in both estimates. This margin of error may or may not be factored in when performing the initial resource assignments, but it has to be accounted for while undertaking capacity planning at both the project and program levels. Not only do calculations have to be performed at the project levels, but also at program levels so as to adequately estimate program-level risks. By changing the way the future demand is calculated and by using the magnitude of project tasks as an additional input, managers can perform resource allocation more effectively and resolve these conflicts before they become full-blown program risks.

Efficient Program Time-Tracking

Assuming that the initial resource supply has been calculated correctly, the real availability of personnel is determined by the projects and program level. An employee working on multiple projects should have the ability to track their inter-project usage statistics. Unfortunately, this is not often the case, even though organizations have deployed various resource tracking systems and rigorously track hours at the project and development life-cycle phase levels. A typical employee only receives guidance on the percentage of time allotted and the overall number of hours the platform has budgeted for the project in question; no consideration is given to the inter-project utilization. In order to perform program-level time-tracking, systems and processes that can track resource usage both at the

Software Defense Application

DoD program managers are increasingly being tasked with managing complex software programs. As such, the number of multi-project programs in the manager's portfolio is not only increasing but constantly changing—a challenge accompanied by a set of unique resource allocation problems not typically encountered in single-project programs. Unfortunately, the program management literature has tended to treat programs as if they are all composed of a single project, thus ignoring these challenges. The objective of this article is to address the challenges faced by program managers in DoD software development programs as they allocate resources across multiple projects. This article will be especially useful to DoD program managers who manage portfolios of projects, project managers who work in multi-project programs, and software engineers and analysts who support programs in the DoD software community.

project and program-level must be implemented. This may be as simple using a spreadsheet or a more integrated enterprise-resource planning system.

Program-Level Task Tracking

A specific project can have many tasks. Therefore, it is necessary to track the tasks at the different levels, project and program alike. Project managers typically conduct task-level resource tracking, but others involved with the program rarely get that opportunity. The program manager is usually conditioned to report on the resource pools at the summary level and does not normally request information about them on the task level. While the program manager rarely has the need, both the project and program managers should be equally aware of the task requirements and tracking to be undertaken (down to the task level) to avoid unnecessary resource conflicts.

As for accounting for different types of outcomes, program/project evaluation and review technique (also known simply as PERT) formulas can become an effective tool to account for variances in resource estimates at the program. For all resources assigned to a task:

- Effort remaining on a task per project = Σ Estimate of Hours to Completion = Σ (Pessimistic* + 4 x Most likely* + Optimistic*) ÷ 6

  ** Estimated remaining hours for any task.*

- The total effort remaining for a project resource at the program level = Σ Effort remaining on all project tasks.
- The total remaining effort for a resource = Σ Efforts on all projects in the program.

It should be noted that calculating remaining work estimates by resource, project, and program is a continuous exercise that should be included as a standard operating procedure whenever earned value analysis is performed on an ad-hoc basis. It may add some incremental effort to the program, but creating a task-level feedback loop ensures that both project and program managers know the true working capacity of their resource pools.

Program Resource Critical Path Analysis (CPA)

The classic definition for a critical path is the sequence of project network activities that add up to the longest overall duration in a project. CPA is a powerful tool that uses mathematical algorithms to schedule complex project activities. It is designed to provide visibility into potential project resource issues so as to develop risk-mitigation strategies.

Extending the critical path concept to a set of related projects is problematic to a lot of programs. This is because while the application of CPA in a single project is well understood, the ability to extend this concept to multi-project programs is challenging. This is because performing a CPA over multiple projects is not a simple activity to complete. It requires the ability to identify the critical path for each project at a low enough task level, a willingness to calculate the true effort for each resource assigned on the critical task, and the coordination of resources for critical path tasks across the program.

The coordination of resources for critical path tasks across the program is paramount to the success of program CPA. A minor oversight in coordination could lead to cascading resource risks across the entire program. One simple yet effective method is the frequency litmus test approach to assessing risks across a program's critical path. It states that a resource cannot be assigned to more than two critical path tasks in the same program. Our empirical observations confirm that most technical resources begin to lose focus when dealing with more than two crises at the same time. And, since tasks on the critical path have the highest probability of generating a true crisis, every effort must be made to avoid situations where a key resource—on critical path tasks—faces multiple crises simultaneously.

Conclusion

The blinding pace of IT innovation and the growing complexity of DoD requirements has led to more challenging multi-project programs. Program managers are charged with managing programs that are more complex, thus requiring intra-project and inter-project resource utilization management. The ability to track and effectively utilize these resources, however, requires effective resource allocation methods.

While the field of project management has successfully developed techniques for managing intra-project resource allocation, its application in multi-project programs is still inadequate. It requires diligent project and program management skills and practices beyond those traditionally required in single-project environments. In addition, it requires the adaptation of best practices—including determining program-level true capacity, efficient program time tracking, program-level task tracking, and program resource-critical CPA. Finally, a concerted effort is required to integrate enterprise-level resource allocation and management tools for program management so as to enable the intra-project and inter-project resource allocation. Without this, effective program management is likely to fail.◆

Note

1. There has been significant analysis and discussion of program and project failures, including the Standish Group's Chaos Report (see the 1995 incarnation here: <www.projectsmart.co.uk/docs/chaos-report.pdf>), the Bull Report (1998), and the KPMG Canada Study (1997).

Additional Resources

1. Bainey, Kenneth R. *Integrated IT Project Management: A Model-Centric Approach*. Norwood, MA: Artech House, 2004.
2. Chin, Gary. *Agile Project Management: How to Succeed in the Face of Changing Project Requirements*. New York: AMACOM/American Management Association, 2003.
3. Gray, Clifford, and Erik Larson. *Project Management: The Managerial Process*. 2nd ed. New York: McGraw-Hill/Irwin, 2002.
4. Hill, Gerard M. *The Complete Project Management Office Handbook*. 2nd ed. Boca Raton, FL: Auerbach Publications, 2007.
5. Schwindt, Christopher. *Resource Allocation in Project Management*. Berlin/Heidelberg: Springer-Verlag, 2005.

About the Authors

Edward B. Lari is a senior principal systems engineer at General Dynamics, where his professional experience includes the application of operations research models in support of large-scale DoD resource allocation problems. He is a doctoral candidate in systems engineering at George Washington University (GWU), has a master's degree in manufacturing systems engineering from the Rensselaer Polytechnic Institute, and a bachelor's degree in manufacturing engineering from Central State University (Ohio).

5311-D Columbia RD
Columbia, MD 21044
Phone: (301) 537-7652
E-mail: laried@gwu.edu

Jeffrey Beach, D.Sc., is the head of the Structures and Composites Department at the Carderock Division of the Naval Surface Warfare Center, where he has worked since 1969. He earned his bachelor's and master's degrees in aerospace engineering from the University of Maryland, and his doctorate in engineering management and systems engineering from GWU.

School of Engineering and Applied Science
The George Washington University
Dept. of Engineering Management and Systems Engineering
20101 Academic WY
STE 227-A
Ashburn, VA 20147
Phone: (202) 994-5450
E-mail: beachje@gwu.edu

Thomas A. Mazzuchi, D.Sc., is chair of the Department of Engineering Management and Systems Engineering at GWU, where he is also professor of both operations research and engineering management. Mazzuchi earned his doctoral and master's degrees in operations research at GWU, and a bachelor's degree in mathematics at Gettysburg College.

School of Engineering and Applied Science
The George Washington University
Dept. of Engineering Management and Systems Engineering
1776 G ST NW
STE 101
Washington, D.C. 20052
Phone: (202) 994-7424
E-mail: mazzu@gwu.edu

Shahram Sarkani, Ph.D., is faculty adviser and head of GWU's engineering management and systems engineering off-campus programs office. He has served as a professor of engineering management and systems engineering since 1999, and is the founder and director (since 1993) of GWU's Laboratory for Infrastructure Safety and Reliability. Sarkani earned his doctorate in civil engineering from Rice University, and his master's and bachelor's degrees in civil engineering from Louisiana State University.

School of Engineering and Applied Science
The George Washington University
Dept. of Engineering Management and Systems Engineering
2600 Michelson DR
STE 750
Irvine, CA 92612
Phone: (949) 724-9695
E-mail: sarkani@gwu.edu

Optimizing Myers-Briggs Type Indicator Training: Practical Applications

Dr. Jennifer Tucker
Otto Kroeger Associates

While many in the DoD systems development community have been exposed to the Myers-Briggs Type Indicator (MBTI)® assessment, many acknowledge not actively applying its insights to their work. Ultimately, because every system begins and ends with the human mind, the cognitive theory that underlies the MBTI is directly applicable to development work and management. This article summarizes practical ways to apply the MBTI to project and systems management.

Most experienced hands in the DoD system development community have encountered the MBTI assessment. A common tool for better understanding communication, leadership, and teams, the MBTI has been administered to millions of people over more than three decades. That means a lot of workshops, a lot of money, and a lot of hours.

Despite this, trainers frequently find themselves leading MBTI workshops where most participants have taken the assessment before, but where few remember the preference scales or their personal type preferences. Even fewer are able to describe how they have used type to better manage themselves, others, and the projects they lead. A recent participant at a Defense Acquisition University workshop I facilitated captured it best: "DoD implementation of the MBTI has historically been, at best, suboptimal."

This article reviews practical and concrete methods for applying the MBTI assessment and its underlying theory of psychological type. An applied understanding of psychological type to solve problems helps build better systems and can improve the optimization and ROI on MBTI training programs.

Psychological Type

The MBTI assessment is based on the work of Carl Jung, a Swiss psychiatrist who developed psychological type, one of the most comprehensive theories explaining human personality. The psychological type model proposes that we each have in-born preferences for how we get our energy, go about gathering information and making decisions, and generally orient ourselves to our world. The theory is captured in the four scales of MBTI assessment. Each scale represents a dichotomy of preferences. Just as you have a preference for writing with your right or left hand, type theory asserts that you have a preference for one of two sides of each of the four scales. Figure 1 summarizes the scales and the preferences, with the center items being the dichotomies, also known as *preference pairs*.

Too often, MBTI workshops explain the scales and help people identify their preferences on the scales, but then reach the end of the time allotted. This leaves little to no time to explicitly relate the principles of psychological type to the actual problems of project management and systems development. Another common occurrence is a workshop that focuses purely on relationship issues, but without an explicit connection between those relationships and the achievement of project goals.

> *"An applied understanding of psychological type to solve problems helps build better systems and can improve the optimization and ROI on MBTI training programs."*

From Individual Preferences to Project Performance

The links between psychological type and project performance are strong. The way our brains function drives how we engage with projects. When people come together on a systems project, their preferences play out together at the project level. In fact, psychological type preferences can often be detected as specific project-level patterns that are either actively supporting or threatening success. As described in [2], projects are often at the most basic level:

- Externally facing or internally facing (aligned with either Extraversion or Introversion)
- Fact driven or possibilities driven (aligned with either Sensing or Intuition)
- Product focused or customer focused (aligned with either Thinking or Feeling)
- Structured or emergent (aligned with either Judging or Perceiving)

What follows is an example of how individual psychological preferences play out at a systems level.

At a personal level, Judging or Perceiving relates to whether we prefer showing the world our decisions or our data gathering. Judgers tend to prefer closure, structure, and decisiveness in public, and keep their data gathering internal. Perceivers tend to prefer openness, flexibility, and adaptability—and keep their decisions internal. At a systems level, this is often reflected in how project teams manage changing requirements and the systems development process itself. Teams that emphasize Judging tend to work to minimize uncertainty, often attempting to lock in requirements early, consistent with a Waterfall development world view. Teams that emphasize Perceiving tend to take a more emergent adaptive approach to development, evolving designs based on the learning done over time.

The best systems development approaches blend closure and adaptation, characteristics of Judging and Perceiving. Examining the processes followed in well-constructed iterative, adaptive, and Spiral software development methodologies reveal this balance. These processes reflect fluidity in moving between activities that gather information (eliciting requirements, storyboarding), focus on decisions (deliverables sign-off, product releases), and those that blend the two (prototyping to both expose new needs and determine desired directions; testing to both expose bugs and resolve them).

At the individual level, the goal is to

® MBTI, Myers-Briggs, and the Myers-Briggs Type Indicator are registered trademarks of the MBTI Trust, Inc.

identify preferences to arrive at one's four-letter personality type: ESTJ for Extraversion, Sensing, Thinking, Judgment and INFP for Introversion, Intuition, Feeling, Perception. A personality type is made up of the four letters that reflect the aggregation of a person's preferences on each scale: For example, I prefer Introversion (I), Intuition (N), Feeling (F), and Judging (J), so my *type* is INFJ. Understanding my type allows me to better manage my personal strengths and blind spots, on a project or in any other environment. At the project level, the goal is to balance all the preferences and their contributions across the project; this is because in the end, each of the eight preferences provides a benefit to a project. By learning to balance each pair of preferences, we can realize the contribution of each and mitigate the risk of over-emphasizing one at the expense of the other.

Table 1 summarizes the patterns I have seen on projects over the course of the past decade, where the overemphasis of one preference at the detriment of the other has translated into concrete risks on real projects. More examples of these are covered in both [2] and [3].

Type as a Problem Solving and Decision-Making Model

Much of what leaders and teams do each day, including the tasks that support systems development, consist of two primary activities: taking in information and making decisions based on that data. As such, data gathering and decision-making are key components of planning and problem solving.

The second and third scales of the psychological type model, Sensing and Intuition and Thinking and Feeling, were identified as the mental functions by Carl Jung. Sensing and Intuition are the preferences that make up the perceiving function, or how we prefer to gather information. Thinking and Feeling are the preferences that make up the judging function, or how we prefer to make decisions. Together, these four preferences create a practical and easy-to-apply decision-making and problem solving model.

Table 2 (see the following page) represents the components of an all-function model for problem solving and decision-making. There are three steps to using this model:

1. Explicitly state the specific problem being faced, need to be filled, or the decision to be made. This can be harder than it sounds. Sensing groups tend to dive into the history and the details, and must integrate these into a single statement that captures the building blocks. Intuitive groups tend to see everything as related, and must work to develop a statement that is concrete and specific enough to work with in a meaningful way.
2. Next, for that need or problem statement or decision to be made, ask the questions in Table 2. Most teams find it helpful to work through the questions in order. Block the time so that the group reserves an even amount of time for each of the four preferences.
3. Finally, identify the decisions, action items, and next steps that are suggested by the discussion, and craft a plan for implementing and communicating these.

I have seen groups effectively self-facilitate this process in a one- to two-hour meeting. With some focus, these groups start to solve problems and make decisions that can have a genuine impact on project success. Through this process, the team also generally discovers which of the preferences come more easily to the group. Answering the questions linked to preferences that are on the radar will generally yield more familiar conversations (i.e., the content will likely already be on the group's view screen). For the preferences that come less easily to the group, the questions are likely to reveal new and fresh information that has not yet been previously considered. This is content that may have previously fallen under the radar; the associated type preference, therefore, might benefit from some additional attention.

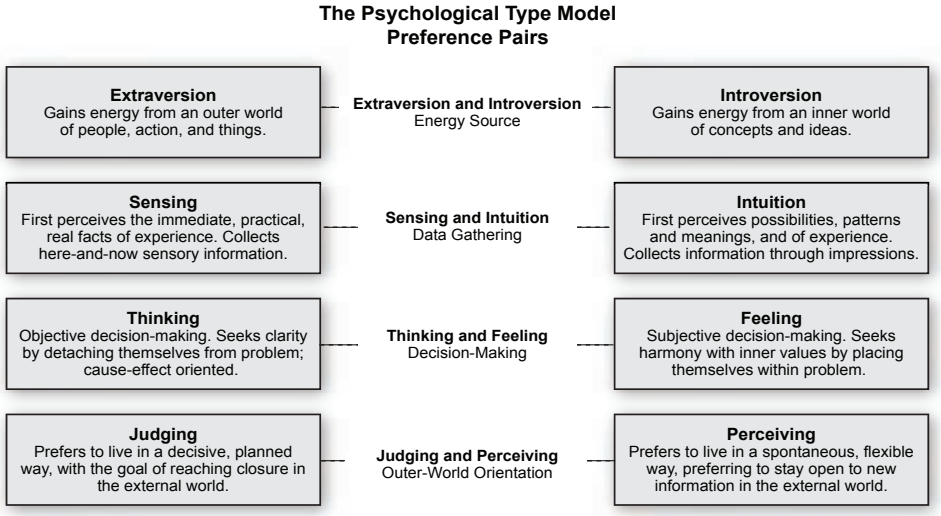


Figure 1: *MBTI Assessment Preference Scales* [1]

Type and Project Planning

This same all-function model can be

Table 1: *Preferences Out of Balance Lead to Risks*

MBTI Preference	Result of Overemphasis
Extraversion	Scope creep, if too much discussion leads to more action items, and possibilities spoken aloud are interpreted as directions and decisions.
Introversion	A lack of scope clarity and alignment, if people assume that others know the decisions and direction when (in fact) they do not.
Sensing	More time tracking than doing, if the team focuses too heavily on the granularity and specifics of schedule and task elements.
Intuition	Slipping timelines, if the concrete realities of task performance take longer than the big-picture visioning anticipated.
Thinking	Lack of user involvement and stakeholder buy-in, if technology-focused specifications overcome people-oriented interests.
Feeling	Lack of tough trade-offs and risk management, if the team works to keep all stakeholders happy and avoid necessary constructive conflict.
Judging	Project completion without project success, if the drive for closure overcomes the benefits of needed discoveries along the way.
Perceiving	Interim success without actual completion, if the pursuit of options leads to exhaustion of time and budget without a final product.

Preference	Effective Questions: "In Making This Decision or Solving This Problem ..."
Sensing	• What is the current situation, as it is? What facts and details describe where we are? • What past experience can we learn from? • What are the important details on which to focus? • What should we keep that works?
Intuition	• What are the patterns or themes across the details and past experience? • What is the big-picture view? What is the vision we want to achieve? • Are there relevant models or concepts to help frame the issue? • What are the possibilities or options? What could we do?
Thinking	• What are the criteria that will determine the best approach? How will we decide? • What are our best alternatives with their respective pros and cons? • What are the most logical solutions? • How will we objectively assess success?
Feeling	• With whom do we need to collaborate and in what ways? • How will the proposed options and goals impact the various people involved or impacted in the situation? • Which approaches will promote maximum acceptance and ownership? • How will we communicate our plan to others?

Table 2: *All-Function Problem-Solving and Decision-Making Model* [1]

applied to project planning. It is widely accepted that project planning done well can save time later, particularly when it is done with an eye to how uncertainty and change will be handled downstream. Given the importance of project planning, it is an attractive activity to consider through the lens of psychological type.

From a psychological type perspective, early planning—because it is by definition future focused—is often positioned under the Intuition preference, with common references to visioning and thinking *outside the box*. Later, when work breakdown structures are underway, the more present and detail-focused Sensing preference may take center stage. When balanced well, all four preferences support effective planning:

- **Sensing:** Provides experience-based data and best practices that can help bridge the present to future vision.
- **Intuition:** Provides future possibilities, patterns, and conceptual models for the future.
- **Thinking:** Provides objective evaluation criteria for decisions and trade-off analysis.
- **Feeling:** Considers how directions and decisions will be perceived by and impact different groups and people.

Table 3 summarizes what project plans can struggle with when the Perceiving and Judging preferences are out of balance, and some steps that can be taken to re-balance each pair.

Table 3: *Equalizing Unbalanced Functions*

Action	When Either Side is Overemphasized		Ways to Balance
Data Gathering	**Sensing.** Plans become an overly detailed compilation of people's anticipated daily activities, and are too specific and all-encompassing to reveal the big-picture and general decision-making criteria.	**Intuition.** Plans become too abstract to be useful, miss the realities that are required to craft schedules and budgets, and are not grounded in concrete data to support daily decision-making.	• For each element of the vision, take the time to break it down into the concrete action and steps that will bring that element to fruition. • Once the details are in place, regularly recheck to affirm alignment with the big picture. • If the project gets too *in the weeds*, take the time to identify root causes, patterns, and brainstorm possibilities. • If the project gets *lost in theory*, work to identify specific tactical steps that can be taken immediately to head in the right direction.
Decision-Making	**Thinking.** Plans leave out the customer's perspective and fail to spend adequate time on change management and communication.	**Feeling**. Plans omit the decision criteria and performance measures that will drive tradeoff decisions and objective measures of progress.	• Develop objective criteria for alternative analysis and trade-off decision-making. • Assign both roles and specific times in the project schedule for stakeholder and customer outreach. • If the project overemphasizes products and problems, stop and take time to focus on how stakeholders are being consulted and included. • If the project is overemphasizing people concerns, ask what needs to be done to refocus on objective decisions.

Balancing Preference Pairs at the Project Level

The discussion now turns to the implementation and communication phases, and how to balance issues like Extraversion and Introversion, as well as Perceiving and Judging, at the project level.

For example, Extraversion and Introversion are useful in understanding one's source of energy at the individual level. At the project level, the preference pair can be useful for determining and monitoring stakeholder and team involvement and communication.

Projects that are out of balance towards Extraversion (too much external focus) can struggle from an over-involvement of stakeholders, leading to difficulties with expectations management and in keeping sensitive or incomplete ideas from being widely released. Projects that are out of balance towards Introversion (too much internal focus) can struggle with a lack of communication, and may fly so far under the radar that they lose upper leadership support.

For balancing Extraversion and Introversion, ask the following questions:

- Who should be involved in what kinds of project decisions and how will they be engaged?
- What is the needed breadth and depth of engagement with different stakeholder groups?
- How will we communicate and maintain connections with team members, senior leaders, and customers?

Judging and Perceiving—which relate to preferences for closure and openness—are invaluable tools at the project level for balancing both attentiveness to meeting milestones and openness to opportunities and contingencies.

Projects that are out of balance with Judging can struggle with coming to closure too early, with concerns about meeting milestones trumping considerations of new information. In these cases, contin-

gency planning is often left off the list, as the project team works to eliminate uncertainty rather than plan for it. Projects that are out of balance with Perceiving struggle with staying open too long, running out of time and resources as different alternatives are explored.

For balancing Judging and Perceiving, ask the following questions:

- What milestones will be set as check-in points, and how will these check-ins occur?
- How do we plan to be flexible? How will we know when a goal or objective no longer makes sense, and how will we regroup when that happens?
- What are some of the contingencies that would trigger a review of the project plan, processes, and proposed products?

Maximizing the Return on Investment

Once they learn about the benefits of applying type to project and systems management, many managers ask whether they should be selecting team members based on preferences. It is tempting to want to take this route; however, because it evaluates preferences—but not skills, knowledge, or abilities—the MBTI assessment is not a valid tool for hiring, team selection, or promotion.

Instead, managers should construct a list of the behaviors and experiences that would ideally be reflected across a project team, or in a specific role that needs filling. The ideas explained in this article can be used to inform that list to select a diverse set of people for the team, or to identify individuals that would bring behaviors, skills, and experiences that are underrepresented. For example, systematically looking across the preferences may highlight skills that are needed, but that may not have been previously valued. The goal is to recruit a group of people that can deliver on those skills, regardless of their actual preferences.

It is never too late to integrate type or the MBTI assessment as a tool in project planning or execution. For example, a mid-point project review utilizing the all-function problem solving model may help reveal patterns in the project's risks that had not been detected, or actions that had not been previously considered to correct course. It can even be used at the end of a development project as the mission changes from development to longer-term operations or deployment. Wherever the project might be in its life, reviewing the benefits of each preference is a great step for continuous improvement. Start with this exercise, which asks some basic questions:

- When considering all eight type preferences, what are we doing well?
- What are we not doing so well?
- What do we need to do differently based on this?

Often, the answers will point to opportunities for greater balance across the preferences, to both maximize strengths and fill holes.

Systems development and IT professionals are frequently criticized for focusing on the benefits of a specific software or process tool, rather than focusing on the customer's functional need or problem. Too often, professionals marketing the MBTI assessment do the same thing: They market and deploy the tool without making the direct link to a problem or need that the customer actually has. This results in fun workshops and enhanced self-awareness for those who connect with the tool, but may not be the optimal outcome for the system as a whole, given the cultural investment in the MBTI across the DoD.

The reality is that the theory underlying the MBTI is a remarkably flexible one, and can be used as a diagnostic assessment and intervention tool at multiple levels: individual, team, project, and organization. Now that many DoD programs and projects have experience with the MBTI, it is time to take the next step, expanding its application to more deeply understand and address all the thorny challenges of project performance. Doing so will lead to better conversations, better teams, and better systems—one program at a time.◆

Software Defense Application

This article strives to connect relationship management with software development through a tool that many in the DoD are familiar with—but have not yet optimized in project delivery. It provides concrete examples of how the MBTI can be applied to better understand common software development challenges, building on MBTI knowledge that many DoD personnel already have, but are not yet clear on how to apply.

References

1. Rutledge, Hile. *MBTI Introduction Workbook*. Fairfax, VA: Otto Kroeger Associates (OKA), 2006.
2. Tucker, Jennifer. *Introduction to Type and Project Management*. Mountain View, CA: CPP, Inc., 2008.
3. Culp, Gordon L., and Anne Smith. *The Lead Dog Has the Best View*. Reston, VA: ASCE Press, 2005.

About the Author

Jennifer Tucker, Ph.D., is the consulting director of OKA, where she facilitates technical and scientific work groups, conducts leadership and team development workshops, and leads organizational assessments. Her work focuses on applying both personality models and social theories to help practically reframe and navigate the complexities of scientific and technological systems and teams. Tucker holds a bachelor's degree in environmental science, a master's degree in management, and a doctorate in science and technology studies. She is the author of the booklet "Introduction to Type and Project Management" and has been a certified Project Management Professional since 2005.

OKA
3605 Chain Bridge RD
Fairfax, VA 22030
Phone: (703) 591-6284
E-mail: jtucker@typetalk.com

Web Sites

The 2010 Software Best Practices Webinar Series

www.itmpi.org/webinars

The IT Metrics and Productivity Institute sponsors a series of Webinars dedicated to improving the practice and management of software development and maintenance worldwide. All live Webinars are free, have been accredited by the Project Management Institute, and earn participants professional development units. Each features an expert speaker who has extensively researched and successfully applied best practice principles to the development and maintenance of software. CrossTalk mainstay Capers Jones—along with other past authors Christof Ebert, Donald J. Reifer, Herb Krasner, Ian Brown, Robert Charette, David Herron, Dan Galorath, Arlene Minkiewicz, David Garmus, and Neil Potter—are scheduled to hold Webinars this year.

DoD Acquisition Workforce Career Management

www.dau.mil/workforce

After reading *Influencing Software Competencies Across the DoD Acquisition Workforce*, why not go to the Web site focused on that group? You can read about the Defense Acquisition Workforce Improvement Act and its latest changes, view a detailed catalog of the DAU's classroom and online learning opportunities, and learn about what specific entities (Army, Navy, Air Force, and civilian employee services) are doing in regards to acquisition workforce support. Also available are the latest memos from DoD leadership regarding strategic goals, education, career advancement, and certification strategies.

How Did the Originators of the Agile Manifesto Turn from Technology Leaders to Leaders of a Cultural Change?

www.infoq.com/articles/manifesto-originators

If you liked Dr. Orit Hazzan and Dr. Tali Seger's article on self-efficacy in software, CrossTalk recommends their InfoQueue exploration of the Agile Manifesto—and its creators. This time with co-author Gil Luria, Hazzan and Seger share in-depth interviews with 12 of the 17 originators of the Agile Manifesto, searching for influences and exploring how technology-driven forces led to the cultural change introduced by the Agile approach. Interviews with the framers go back as far as childhood in search of influences, explore their early professional lives in technology and software, gain differing perspectives from that famed February 2001 meeting, and explore how their professional roles have changed in the nine years since.

Human Asset Management

Martin Allen
Independent Software Consultant

Widely recognized as the "father of modern economics," Adam Smith's seminal book, "The Wealth of Nations," included both tangible assets (machines, buildings, and land) and humans as essential wealth-generating resources. Our present high-technology industries are eager to invest in and protect their tangible assets (such as computer networks), but the modern accountancy paradigm forces the view of an employee as merely a cost. Humans are indeed primary assets, and this article provides guidance in assessing, utilizing, and enhancing their value.

Adam Smith was both influential and controversial in his inclusion of humans, or labor, as a major contribution to the wealth of a nation. Then again, Smith was not the only visionary in 18th century Scotland to acknowledge the human contribution to business and economics. With the Industrial Revolution in full flow, the common worker was perceived by industrialists as a replaceable cog in a machine, there to turn a profit and to be exploited. Cotton mill owner and social reformer Robert Owen would break that mold. On the banks of the River Clyde (near Glasgow) stands the one-time cotton mill town of New Lanark [1]. As testimony to its achievements, the mill operated from 1786 until 1968 and is now preserved as a World Heritage site. What distinguished the mill-owner from others at that time was the manner in which he nurtured the workforce: Having decent homes for workers, schools for their children, and cooperative shops delivering goods at a reasonable value contributed to the town's financial and communal success. For Owen, this was not philanthropy; it was the self-interest of capitalism, spiked with a social conscience. As clearly as the River Clyde was needed to turn the wheels of the mill machines, Owen recognized that people were needed to work the looms that weave the cloth. The end-result was employees who worked harder and better—and created a higher-quality final product.

Human Assets

Adam Smith and I share more than nationality; we both believe that humans are key assets in the pursuit of economically favorable results. For more years than I care to admit to, I have held a strong conviction that substantial and sustainable advances in industrial-scale software engineering will not come from new tools or programming languages; they will not emerge from the hyperbole surrounding heavyweight or lightweight processes; they will not come from accruing quality badges or collecting metrics; and they will not arrive via a modeling notation (such as the Unified Modeling Language). The potential for improvement is the most that various software tools, techniques, and initiatives can ever deliver. Substantive progress has been—and will continue to be—a direct consequence of employing software professionals and providing them with a suitable environment in which to operate. Competent personnel are an organization's pivotal assets.

> *"... people aspects seem to dominate the most expensive project disasters."*

Having an appropriately trained software workforce has a double-edged effect. Educated teams focus on productive work; shared knowledge and experience give them the cohesion with which to address the intellectual tasks that comprise engineering. Equally significant is that teams spend less time evaluating and correcting substandard software artifacts simply because these are produced in negligible quantities by competent engineers.

While there is a widespread fallacy that technical issues are the primary source of project woes, people aspects seem to dominate the most expensive project disasters. Without a doubt, the majority of perceived difficulties are simply symptomatic of intrinsic people factors (*politics* is a popular euphemism for these). To quote from the influential book, "Peopleware":

> For the overwhelming majority of the bankrupt projects we studied, there was not a single technological issue to explain the failure ... The major problems of our work are not so much technological as sociological in nature. [2]

Particularly in knowledge economies, the success or failure of technology programs hinges on assessing capabilities and recognizing the needs of the engineering teams. To this end, I have generated seven human-centric rules for software development organizations to adhere to:

- **Rule 1:** The main causal factors of project success, mediocrity, or failure should be recognized as human and organizational, not technological.
- **Rule 2:** Professionalism and software engineering competence should be assessed objectively and encouraged proactively by senior management.
- **Rule 3:** The number and seniority of software professionals employed within an organization should be commensurate with the magnitude and criticality of the required software systems.
- **Rule 4:** Organizations should provide an environment conducive to the intellectual task.
- **Rule 5:** Management should recognize its primary functions are to attract, motivate, facilitate, and retain talent. Teams should be given an identity, a vision, and quality goals.
- **Rule 6:** Teams should be organized with respect to member strengths and competencies.
- **Rule 7:** Dependable sources of knowledge should be provided in the form of textbooks and training materials.

Readers familiar with my May/June 2009 CrossTalk article will recognize two of these rules. Attention was also drawn to three essential ingredients of software development called the 3Ps: people, products, and processes [3].

Human Asset Evaluation

Not all assets are of equal value; so too can be said of human assets. The process of evaluation for most assets is based fundamentally on attributes and objective criteria, with a variable dash of subjectivity

on top. Houses with identical specifications and locations (objective) can have different valuations due to the varying attraction of the décor (subjective). Likewise, diamonds of the same number of carats (objective) can have different valuations due to their luster (subjective). When assessing the value of human assets, we can follow a similar scheme.

The main objective criteria for evaluating professional (software) engineers are:

- Qualifications/education.
- Formal training.
- Experience.

If necessary, qualifiers can be used to turn these into quantitative criteria—consider the applicability, duration, and timeliness of each criterion. For example, a relevant degree is very valuable; if my experience is typical, this has shaped an engineer's aptitude from an early age.

Subjective criteria, sometimes referred to as *soft* skills, may include:

- Communication ability.
- Attitude/commitment.
- Adaptability.
- Assertiveness.

Decent soft skills are particularly relevant where team interaction and influence are important.

One may expect the formality of an employee evaluation scheme to vary with respect to the industry and the criticality of the application. With safety-related software systems in Europe, the range of individual and team competence is unacceptably wide. In recent years, I have had the pleasure of working alongside competent professionals with the requisite qualifications, training, and experience. I have also worked in critical environments where, clearly, the engineers lacked even basic skills. Some years ago, I was involved in an initiative to introduce a competence assessment scheme into a safety-related industry sector. A significant number of managers and engineers were unwilling to take part in the scheme until the objective criteria had been replaced by subjective criteria. Those same people would probably balk at the idea of being passengers on an airplane flown by a pilot who was not trained, but who had a pleasant voice over the speaker. Unfortunate as it seems, adopting formal employee evaluation or competence assessment schemes is like diet and exercise—we need it to stay fit and healthy, but doing the things that are best for us is not always easy. For safety-related systems, the British Health and Safety Executive has published guidelines for introducing a competence management system [4].

Could the certification of software engineering professionals be the answer (or part of the answer) in establishing competence? As is often the case when challenged to answer a software engineering question, we rely on respected sources of knowledge such as CROSSTALK. Perhaps, surprisingly, there is a lack of information on professional certification; the obvious conclusion is that certification is at least an unpopular subject, or even taboo. Without a doubt, there are leading academics and practicing professionals who are concerned about the efficacy of certification programs. The core of the IEEE's Certified Software Development Associate and Professional efforts [5], the Software Engineering Body of Knowledge, has been criticized; for example, its ability to encompass all application domains is questionable. However, my own experiences (in various industry sectors in Europe) show a multitude of people, particularly in technical leadership roles, who would be fearful of successful certification initiatives that could expose their shallow grasp of a deep discipline.

Perhaps there are less formal and more palatable ways of assessing an engineer's competence. For instance, I have assembled a portfolio from the many projects I have worked on. This collection includes samples of requirements specifications, architectural designs, test specifications, process definitions, presentations, lists of technical books read and owned, etc. This gives me the ability to show someone the level of my experience and ability; yet, having a portfolio is far from common in our industries and it receives very mixed reactions. As DeMarco and Lister suggest, "It would be ludicrous to think of hiring a juggler without first seeing him perform" [2].

On the topic of hiring competent staff, professionals must be dismayed at the high proportion of job advertisements, over a substantial period, focused on low-caliber skills. For instance, experiences with a specific programming language or a particular requirements management or design tool are often cited as essential skills. With regard to requirements, the major skill is always in the specification: Tool proficiency can inject quality into requirements management, not the specification thereof. A skilled engineer will be trained to specify atomic, consistent, structured, and testable requirements; Wilson provides a synopsis on requirements specification in [6]. Similarly with design tools, and to quote Grady Booch, "CASE [Computer Assisted Software Engineering] tools have allowed merely bad designers to produce bad designs more quickly" [7]. The absolute fixation by whole industry sectors on programming language experience is a continuing embarrassment. There is an ever-present concern in our profession that the wrong categories of skills are encouraged and valued.

Judging from the feedback I received from my previous article, the most contentious areas were the rules and the section dealing with people—specifically, the contrasting behavior between professionals and amateurs. For a comparison, see Table 1.

We can take one of the divergences between these groups of people and analyze it further. Consider a sliding scale with art at one end and science at the other. If, by a process of task analysis, we conclude that the creative nature of software engineering and its resilience to practical mathematical proof places it nearer art than sci-

Table 1: *Characteristics of Software Professionals versus Amateurs*

The Professional Practitioner	The Amateur or Hobbyist
Views the overall task as an engineering discipline.	Describes the overall task as an art or craft.
Promotes a holistic, life-cycle view.	Holds an implementation, coding bias.
Places emphasis on the application or problem domain, and presents architectural solutions.	Places emphasis on the technical detail of the solution domain to the detriment of the customer or user.
Learns principally from published engineering literature.	Learns principally by emulating colleagues.
Encourages compliance with industry standards.	Prefers improvised, local procedures.
Employs quality criteria to manage projects.	Manages projects via schedule alone.
Conveys an outward, discipline focus.	Conveys an inward, project focus.
Exhibits a balanced approach to risk.	Adopts a naïve approach to risk.

ence, we must pause for thought. Good literature is founded on the discipline of strict linguistic standards (e.g., punctuation, spelling, and grammar), whereas music is founded on structures for tone, rhythm, and notation. History has proven, therefore, that discipline has released creativity, not stifled it: Discipline is as elemental to an artist, writer, or musician as it is to an engineer.

When a student receives a classical education in software engineering, this indoctrinates a view of, and an approach to, the discipline that is not just different from common practices, perceptions, and mythology—it is diametrically opposed. The gap between professional and amateur is not a gap, it is a chasm. Therefore, in comparative arithmetic terms, it is ludicrous to assume that 10 untrained personnel can perform even the work of a solitary professional. Perhaps this accounts for the 10 to 1 productivity ratio recorded as long ago as 1975 by Frederick P. Brooks in "The Mythical Man-Month" [8].

In 1980, I was a new graduate working in the British defense industry. I was approached by a concerned manager who observed, "You appear to be faltering and are not producing code as quickly as your peers." No one amongst this large office of software developers had ever witnessed a qualified and trained softie ratifying and specifying requirements, devising software architecture, designing the software, defining test cases and recording test results, as well as generating robust code. Wind the clock forward almost three decades and, in a high-dependability environment, our team was tasked to review multiple software requirements specifications produced by untrained personnel. Even with the availability of practical guidelines, the authors had produced worthless specifications. Combined with the worst that a bottom-up, functional software architecture has to offer, the project was in an undesirable state. The real problem in such a scenario is that, again, a group of untrained people cannot match the actual productivity of one professional. Equally, the majority will dominate proceedings and a solitary professional will toil to correct the substantial but substandard output of 10 untrained employees.

Lean Engineering is a populist topic, although it is well-documented that production-oriented techniques do not transfer readily into a (software) development environment—also known as the *make a cheeseburger, sell a cheeseburger* mentality discussed in [2]. Nonetheless, one of the principles of Lean is the reduction of waste in a production line. In terms of waste, having non-professional software personnel producing substandard artifacts is analogous to having an untrained team preparing and then selling raw, frozen burgers on a bun, with or without the cheese. In order to rectify this and similar waste issues, an organization may choose to assess the capabilities and training needs of project personnel—or choose to assess the competence of the management team that appointed and tasked untrained personnel in the first instance.

If the economic aspects of the software engineering life cycle were ever to be modeled, the most significant variables in the equation would reflect the human knowledge and experience. Then again, the life cycle has been modeled and the people capability attributes are the most significant. Barry Boehm identified this truism as early as the 1980s, and it is captured in COCOMO [9].

There are already significant clues here as to how to assess the value or competence of software engineering personnel. However, if your organization is still searching for a magical productivity enhancer, then look to laetrile[1] for the futility of searching for wonder cures, or "easy technological non-solutions" as described fully in chapter 6 of [2] and again in Brooks' "No Silver Bullet" chapter in [8].

Human Asset Enhancement

As we know, many assets (like cars) depreciate in value, while others (like real estate) increase in value—so we maintain, build on, and insure against loss those appreciating investments. Humans increase in worth through enhanced knowledge or experience and, likewise, an organization is prudent to invest time and money on its most precious people assets as well as to guard against the mishap of their loss. In our competitive technology-driven markets, companies striving to be the best have to attract, nurture, and retain high-caliber engineers.

I remember an organization that purchased each member of its administrative personnel a top-of-the-line PC and smart laser printer. What happened? The engineers stopped squabbling over their rented, one-between-five, basic machines—and looked on enviously. In this case, the organization showed a blatant disregard for its engineers. After this, many resumes were typed into those rented computers, including my own. When an organization equips all of its people adequately, provides an environment conducive to the intellectual task of technological development, and supports personnel growth, it in turn maintains, enhances, and protects human asset value. Engineers are people, and people need to feel their contributions are valued.

As indicated earlier, staff competence assessment schemes are often viewed negatively—but suppose such assessments were shown to link directly to an organization's development of, and investment in, its people. Therefore, the most effective teams can be organized on the objective basis of competence rather than on arbitrary and subjective opinions. As initial reluctance gives way to synergy, people will gel into strong teams.

Having assembled a strong team of competent professionals, how should it be organized? Brooks gives sound advice in his "The Surgical Team" [8] chapter. A surgeon and his surgical assistant perform an operation, while supported by nurses, an anesthesiologist, and administrative staff. This arrangement compares favorably with the roles of software architect, software manager, programmers, testers, and an administrator[2]. Consider the difference in the structure and formalism within different groups of musicians. A random collection of musicians can meet for a jam session; without proper direction or sheet music, the small group can still produce some decent, improvised sounds. Similarly, a small band of jazz musicians can, given reasonable levels of competence and practice, entertain an appreciative audience. However, to reach the excellence of a large professional orchestra is a monumental challenge in systematic coordination—the conductor on the rostrum and the sheet music are not just for show. For technical teams developing software-intensive systems of systems, the analogy is clear: Systematic and formal coordination is essential.

Software Defense Application

Readers will benefit from this defense software industry insider's focus on the proven, primary influence on software project success or failure: the combined capabilities of development team members. As for return on investment, enhancing team capability is unique in offering potential productivity gain factors of up to 10 times. Increasing team capability has the dual effect of improving the quality of software artifacts and reducing waste.

Having assembled a strong team of competent professionals, how should it be managed? Consider an environment where managers (verbally) whip teams to meet implausible milestones; where engineers are forced to cut corners and undermine basic quality criteria; where training is regarded as an unnecessary expense, and reading a book is considered a waste of company time; where professional opinions are most unwelcome and people are expected to just get on with the job; where political mendacity is a substitute for competence; where knowledge is wielded like a bludgeon with people herded into pens like cattle; in an office where it is too hot, cold, or noisy for anyone to function efficiently. In contrast, imagine an environment where people are encouraged to design outstanding products; where teams and individuals are challenged to excel; an office with shelves groaning from the weight of books by software gurus; where training is viewed as a necessity; where professionalism is a given; where diverse opinions are seen as the balance in an open political culture; where personnel are provided with adequate tools and comfort; and a place where the organization and staff have mutual goals and aspirations. In summary, it is as easy to obtain the least value from your human assets as it is to obtain the most value.

Conclusion

The legacies of Adam Smith and Robert Owen are an important reminder to us that people are at the heart of commercial and social success. In our rapidly changing technological world, it is worth considering their centuries-old wisdom. Perhaps there is an opportunity for our organizations to look again at the value, rather than the cost, of their people assets. When people are viewed truly as vital assets, then investment in them is sure to deliver a mutually beneficial corporate future. This will, in turn, lead to greater customer satisfaction.◆

References

1. "New Lanark – World Heritage Site." <www.newlanark.org>.
2. DeMarco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams*. New York: Dorset House, 1999.
3. Allen, Martin. "From Substandard to Successful Software." CrossTalk May/June 2009 <www.stsc.hill.af.mil/crosstalk/2009/05/0905Allen.pdf>.
4. Health and Safety Executive, The Institution of Electrical Engineers, and The British Computer Society. *Managing Competence for Safety-Related Systems*. 2007 <www.hse.gov.uk/humanfactors/topics/mancomppt1.pdf>.
5. IEEE Computer Society. "Certification and Training for Software Professionals." 2009 <www.computer.org/portal/web/certification>.
6. Wilson, William M. "Writing Effective Natural Language Requirements Specifications." CrossTalk Feb. 1999 <www.stsc.hill.af.mil/crosstalk/1999/02/wilson.pdf>.
7. Booch, Grady. *Object-Oriented Analysis and Design with Applications*. Menlo Park, CA: Addison Wesley, 1994.
8. Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering*. 20th Anniversary Edition. Reading, MA: Addison-Wesley, 1995.
9. Boehm, Barry W. *Software Engineering Economics*. Upper Saddle River, NJ: Prentice Hall PTR, 1981.

Notes

1. Laetrile is an extract from apricot stones, sold in Mexico as a (fraudulent) cure for cancer.
2. The role of software manager here is one of a facilitator, rather than a technical lead. Team composition should also vary with the magnitude and criticality of the task.

About the Author

Martin Allen is a software engineering professional with 30 years experience, mostly in the defense industry in the United Kingdom. He has worked on many successful software-intensive systems for the British Royal Air Force and the Royal Navy. Allen has always had a strong interest in industry standards for the engineering of dependable systems. His other professional interests include risk management, software cost economics, requirements analysis, design methods, and software testing. Allen and his colleagues work on the boundary between the academic research of computer science and the practical application of software engineering.

1 Belfry Walk
Titchfield Common
Hampshire, United Kingdom
PO14 4QD
E-mail: mjallen60@yahoo.co.uk

Grace Murray Hopper: We Still Need You!

During my 15-plus years of writing BACKTALK columns, I have often found myself working a column while traveling. I am usually in one of three conditions: 1) bored, stuck in my hotel, not much to watch on TV; 2) bored, on yet another flight with no movie and a bad book that doesn't hold my attention; or 3) frustrated and bored, sitting in an airport, waiting for a late flight.

This week, I had the pleasure to attend the SIGCSE—Special Interest Group on Computer Science Education—in Milwaukee. Great conference, but, unfortunately, on my way home, the third condition applied. You see, I planned this conference around my spring break. Once I landed in Houston, I was driving to Orlando with my family to visit my parents. We were leaving as soon as I touched down in Houston, so I was in a hurry.

RADM Grace Murray Hopper

The weather in Milwaukee has been mixed all week, so I watched the weather closely—and on this morning we had fog, rain, and 20 mph winds. However, the airline assured me that the incoming and outgoing flights were on time. When I got to the airport, my departing aircraft was 15 minutes early—it was quickly deplaned, cleaned, the old-for-new luggage exchanged, and the boarding door was opened. All was right with the world.

One problem: There was no flight crew. I found out that while the plane itself had come in from decent Midwest weather, our flight crew was coming in from Newark where, apparently, Noah was floating down the main runway, picking up small animals left and right.

So unfortunately, while all the "hardware" was ready, something critical was missing—the human components.

At one of the SIGCSE workshops, we discussed Massive Parallelism: The analogy was that if one person can complete a jigsaw puzzle in two hours, two people could possibly do it in one. Maybe three could do it in 40 minutes. However, if you put the entire class of 20 students to work on it, it would probably take three or four hours. Human beings do not parallelize well. One of the instructors made a comment that Moore's Law applies to hardware only—and while one could argue that it takes 1.8 years to double software capacity, it takes 18 years to double software engineering capabilities. It's hard to significantly increase human capabilities. My feeling is that as newer and newer skills and languages and technologies come, older knowledge and associated skills decrease. And, given that most of us support the DoD in some way, we are sometimes working with technology that was developed and fielded *many* years ago. Face it: We *need* "legacy" skills to keep older systems up and running.

I have been involved in education and training for more than 30 years. I have taught all the latest trends and methodologies and processes as they develop and evolve. However, I find myself having to make a deep, dark confession: I am *still* teaching COBOL. In fact, I am teaching it in college this very semester. Why? Because there is still a real need for it in the "real world." In fact, there is a growing need! Mature and experienced developers (that's a nice way of saying "the old folks") are retiring, and large companies that have heavy investments in older languages over the years need new developers that can maintain and even expand this older code.

What happens when the current (and aging) pool of developers of legacy code dwindles, and fewer and fewer newly graduated developers know how to maintain older systems? Well, those who know (or are willing to learn) the older and more "traditional" languages will be in demand. Like that old Broadway song says, "everything old is new again".

While working on this column, I was buying an online ticket. The Web site kept rejecting my data for "unspecified reasons." Eventually, I was connected to one of their "application specialists" (short for developers) who said that the problem was that my house address contains the fraction "1/2", and their application could not handle the "/". I jokingly told the developer "Oh, storing the number in a PIC 9, not a PIC X, huh?" He laughed—and asked if I was available for some contract work. Seems they have a hard time hiring knowledgeable COBOL developers. If this anecdote—or, for that matter, the name Grace Murray Hopper[1]—doesn't register with you, maybe it's time to look again at those "legacy" languages.

Remember, there are no old languages—just job opportunities.

—David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu

Note

1. To learn more about RADM Hopper (1906-1992)—including her role in COBOL—visit <www.chips.navy.mil/links/grace_hopper/womn.htm>. As stated in a December 1999 CROSSTALK feature called *Influential Men and Women of Software*, she was:

> ... creator of Common Business Oriented Language (COBOL). She was an officer in the Navy who became an Admiral. COBOL came about in the 1950s when the need for higher order languages was seen as a way to increase the productivity of programming computer applications.

Can You BACKTALK?

Here is your chance to make your point without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. These articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery, and should not exceed 750 words.

For more information on how to submit your BACKTALK article, go to <www.stsc.hill.af.mil>.

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737

CrossTalk thanks the above organizations for providing their support.